Integrity Analysis and Coercion in Distributed Systems

By

Mark Hepburn (BEng.)

Submitted in fulfilment of the requirements
for the Degree of Doctor of Philosophy

University of Tasmania (February, 2006)

Declaration

This thesis contains no material which has been accepted for the award of any other higher degree or graduate diploma in any tertiary institution. To the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference has been made in the text of the thesis.

Mark Hepburn

Statement of Authority of Access

Mark Hepburn

To my loving parents,
Peter and Jeanne Hepburn,
who always taught me to think critically.

Abstract

This thesis presents a new approach to modelling the security and integrity of data in distributed and ad-hoc networks of processes. An annotated type based analysis is introduced which ensures that no contamination will occur between data considered trustworthy and data that may have been corrupted.

A method of performing safe run-time coercion of security properties of data is also presented. This is novel because it enables users to perform run-time coercions of data in a manner that may be statically proven safe.

Both plain networks and dynamic (agent-based) networks are considered. These are modelled as systems of first-order and higher-order pi-calculus, respectively. The higher-order system examined introduces a new notion of trustworthiness dependent on the context in which it is typed or executed. This allows programs with malicious intent to be safely executed when it can be demonstrated that no possibility for interaction with other programs, including the host, is possible. A concept of execution context is introduced to perform this analysis.

In addition, annotated type systems with and without sub-typing are described, and subject reduction is shown to hold for all systems considered.

Implementation of the method is demonstrated via type-inference algorithms, and these are shown to be both sound and complete for all systems.

Acknowledgements

This thesis could not have been completed without the constant support and advice of my supervisor Doctor David Wright and associate supervisor Doctor Vishv Malhotra; I owe them both an enormous debt of gratitude.

Many other people also contributed to its successful completion, albeit most not directly; to all the people that follow (in no real order) — thank-you very much from the bottom of my heart.

To all the staff around the department (and now my fellow colleagues), thank-you. A special mention goes to Julian and Nicole, two of the hardest-working and most dedicated people I know (and in Julian's case, the best impromptu public speaker I've seen, especially if given enough time to prepare). Special mention also to Pete Vamplew, my honours supervisor who encouraged me down this path. I am grateful to Professor Young Choi for allowing me time off teaching in order to complete this document.

I must also thank Ian Lewis, my office partner (when he actually turned up) and partner-in-crime — not to mention fellow sufferer — in the first few years, for making it all fun. (What is said when the door is closed, stays behind the door!). Thanks also go to all the fellow post-grad students who have shared my time here, with special mention to Adam Berry, Chris Mitchell (sorry you didn't stick with us, Chris), and honorary post-grad Pauline. A word of encouragement also to Jacco: it isn't easy being the lone theoretical member of an AI-focused department; I hope you can stick it out!

Support did not come solely from inside the school needless to say. I am indebted to the following people for their encouragement, and grateful for their close friendship. To Oli, the best beerday buddy a man could hope for; to Tom, for a friendship that started in Japan, and also the fittest bloke I know; to Uwe, fellow post-grad in the early days and also neighbour; and Bec, for sharing my taste in music (most of it).

Lastly, but certainly not least, for my warmest thanks, love, and deepest gratitude goes to the following: my parents (to whom this dissertation is dedicated); you brought me up to believe in always doing things the right way and to question everything; you are everything I could ask for and more

(as well as for your unconditional love and support, and feeding me towards the end, even though I didn't see you much!); my brother and flat-mate Jonathan, I'll miss you when you leave; to Amy, without whom I likely wouldn't have embarked on this journey in the first place; and to Anita, for constant and unwavering support during the death-march.

Thank-you all, and I hope to see more of you now that this is finished!

This dissertation was typeset using LaTeX and prepared using the Emacs text editor with the AucTeX package, on a Debian GNU/Linux system. All of this software is a joy to use, is of the highest quality, and is freely — in every sense of the word — available to everyone, thanks to the generous work of a cast of thousands. I am indebted to all of those whose contributions made this possible, and would like to extend my thanks to you.

Acknowledgements For The Revised Edition

This is a revised version of the dissertation. I am extremely grateful to my examiners for their highly detailed comments and the amount of time this must have taken, especially when the quality of the first edition quite possibly did not warrant such attention. If I had had such feedback earlier on, I might not have been rewriting this now.

In the roughly 18 months that have elapsed between the first and second submissions of this dissertation, nearly everyone mentioned above has moved on! Chris had left shortly before submission of the first version, and Jacco unfortunately didn't last much longer (leaving me the only theoretical person in the department!).

My supervisor David has left to pursue his own business interests. I wish you well in these, and am very grateful for the time you were still able to spare for advice during this rewrite. Pete Vamplew has taken up a position at Ballarat University, and will be sorely missed. Oli, while not leaving for good, is about to head overseas for a year. Amy is now in Sydney, and my brother in Brisbane.

All of this makes me especially thankful for the constants in my life, and none more so than Anita whose unflagging support remained very welcome and also not a little surprising, given what she has had to put up with!

Lastly, Ian Lewis despite barely turning up at Uni and often appearing hung-over when he did, not only submitted his dissertation a full year after I did but also received his results back shortly after mine, and of course passed with only minor corrections. I may never live this down.

Contents

List of Figures

Chapter 1

Introduction

This thesis examines the issues of static verification of run-time integrity checks and of determining the conglomerate trustedness, with respect to a given policy, of groups of mobile agents with dynamic communication topologies.

The first-order π-calculus is used to develop the theory, which is then also extended to the higher-order π-calculus. In all cases type systems annotated with a boolean algebra over integrity levels provide static guarantees of safety and security. Safe run-time coercion of integrity type annotations is achieved by extending the language with a simple construct that encapsulates a run-time verification check (the details of which are left abstract by parameterising on an oracle), and branching based on its success or failure. Thus, one branch can be parameterised on a trusted name or variable, and the other on an untrusted name or variable. It is thus possible for the type system to determine if the branches are in the wrong position, for example.

1.1 Motivation

One of the most promising directions in computer utilisation and computing research is the concurrent and distributed execution of tasks. Some of the world's most powerful supercomputers are currently (February 2006) clusters of cheap "off-the-shelf" components, usually at a cost of orders of magnitude less than the traditional specialised hardware. This trend also extends to distribution on a global scale, most commonly (at the time of writing) distributed as screensavers that perform CPU-intensive calculations while their host machine is idle; see the `SETI@Home` project (*SETI@home: Search for Extraterrestrial Intelligence at home* (2003), for analysis of radio signals from space) and `distributed.net` (*Distributed.net* (2003), coordinated

brute-force efforts to crack various encryption schemes) for popular examples of this approach. Naturally this need not be the only method of distribution, and research into platforms, languages, network technologies and other enablers is on-going.

The excitement surrounding this trend however is tempered by the dangers of such an approach: for the result of the computation to be trusted, generally all the participants must also be trusted. The dangers do not stop with the direct actors either; the communication networks employed are themselves vulnerable to attack, perhaps by corrupting data (this may incidentally be the result of interference from noise for example, rather than a malicious attack) or even by injecting false data that may be trusted if it is thought to come from a trusted component.

Research in this area is on-going, and many schemes exist to mitigate the potential dangers.

Encryption techniques can provide protection from two angles: first by maintaining data secrecy (which is not a concern of this thesis), and secondly by verifying the integrity of data from a particular sender using digital signatures (Schneier 1993). Formal analysis of protocols is a related issue, assuming the encryption/signature scheme used is unbreakable (or unforgeable) and attempting to detect any weakness in the protocol used (Abadi 1999, 1997).

Another branch of analysis typically aimed at preserving secrecy of information is information flow analysis (Denning 1976; Volpano, Smith and Irvine 1996; Honda, Vasconcelos and Yoshida 2000; Pottier and Conchon 2000; Hennessy and Riely 2000). The general idea is that there are different levels of classification for data — for example in a military setting there might be the classifications "civilian" and "senior personnel only" — information should be allowed to flow from the lower (e.g. "civilian") level to the higher, but not in the other direction.

Many methods exist for *verification* as well; that is, detecting if the code or data received is trustworthy or not. Digital signatures may establish that data has not been corrupted in transit (although not whether it was trustworthy in the first place), checksums provide some ability to check data integrity (against random noise), and so on. In a higher-order context, systems such as proof-carrying code (for example Necula and Lee 1998) can determine if the program is trustworthy or not (the burden of proof is on the program's author). The Java Virtual Machine (Yellin 1995) also does byte-code verification to check the veracity of each instruction.

Broadly, most analyses can be grouped into *static* (that is, performed once, typically at compile-time; examples include protocol analysis and most forms of information flow analysis), and *dynamic* (performed at run-time;

for example digital signatures and byte-code verification). The systems presented in this thesis accommodate both; they are a static analysis, but in a form that facilitates using different methods of run-time verification as appropriate by acting as a framework. The existence of an 'oracle' is assumed; that is, any procedure that is capable of determining the integrity of a piece of code or data at run-time — the precise nature of this oracle is unspecified, and any suitable method would suffice.

An interesting consequence of this approach is that it equips the programmer with tools not just to guarantee separation of data of differing integrity, but also to *interact* with data of a lower integrity on a controlled and safe basis.

The following sections outline the motivation and contributions of the thesis, and review related work.

1.2 The Meta-Language

A meta-language concisely encapsulating the properties of interest is a great help in a formal analysis. The system properties this thesis concerns itself with are concurrency and distribution, possibly involving mobile code. The predominant language in this setting is the π-calculus (Milner 1993). A few examples should illustrate the basic concepts and the issues tackled by the analysis, with a more complete coverage given in Chapter 2.

1.2.1 First-Order π-calculus

The calculus is built on two primitives, communication and concurrency. The latter is expressed most concisely; given two separate processes denoted by P and Q, their concurrent execution is written as P|Q.

The core notion of execution is reduction by communication. Communication occurs between two processes, one sending a message (which may be empty, in other words synchronisation) and the other receiving that message and binding any data that was contained in the message. Communication occurs along named channels, which may represent for example TCP/IP sockets.

As an example, process P may synchronise — communication is a blocking operation — with process Q by sending an empty message on a channel denoted by x, with the output end specified by a bar over the channel name: $\overline{x}$.P | x.Q (note the use of concurrency as well). On synchronisation, this program would reduce to P|Q.

Expanding on this example a little, P may send some additional information, say the number 3. The receiving process Q will then bind the input to a name y. This is also easily expressed in the language: $\overline{x}[3].\mathrm{P} \mid x(y).\mathrm{Q}$, which on communication reduces to $\mathrm{P}|\mathrm{Q}\{3/y\}$, where $\mathrm{Q}\{3/y\}$ denotes the substitution of 3 for all free occurrences of y in Q.

At this point, a little reflection on the safety issues is in order. The dangers would appear to be that the value 3 may itself not be trustworthy (its value may have been compromised somehow), but also that the channel x itself may not be secure. Perhaps it is susceptible to a man-in-the-middle attack (Schneier 1993), or maybe it is just noisy and frequently corrupts data in transit. This obviously has implications for values sent along it; if the channel cannot be trusted, then logically neither can values received from it.

The problem is potentially complicated further by the fact that channels are themselves first-class data in the π-calculus, enabling dynamic topologies. For example, consider the program

$$x(z).\overline{z}.\mathrm{P} \mid \overline{x}[y].\mathrm{Q} \mid y.\mathrm{R}$$

Note that the process $x(z).\overline{z}.\mathrm{P}$ has no mention of the channel y, and thus cannot communicate with $y.\mathrm{R}$. However, it *can* communicate with $\overline{x}[y].\mathrm{Q}$, and in doing so obtains the name y which it then binds to the name z to get $\overline{y}.\mathrm{P}$. Now it may synchronise with $y.\mathrm{R}$.

A common approach to problems such as this is through a type system, perhaps decorated with additional information concerning the integrity values belonging to that type. In the π-calculus the usual representation of types is as a list of types that channels belonging to it may carry. So in the first example, the type of x might be the singleton list (int). If this is annotated, where b and c range over annotations say, the extended type might resemble $(int^c)^b$. As mentioned, there is a relationship between b and c that must be captured.

This thesis chooses to use a boolean algebra for annotations, where trustedness (T) or integrity is the 1 element, and untrustedness (U) the 0 element. The implication that if b is untrusted then so must c be can be concisely expressed by the requirement that $c = b \cdot c$ (so if b is the zero element then $c = 0$ by implication). More complicated dependencies and relationships can also be expressed in the algebra.

A second way that untrusted channels may adversely affect proceedings is by blocking communication, altering synchronicity by taking an inadvertently long time to complete a transmission, and so on. For example, consider the process $x.y.\mathrm{P}$ that synchronises on two channels in succession, then acts like

P. Assume that x is untrusted and y trusted; then the second synchronisation on y will not be able to occur until the first has occurred, which means that it is relying on an untrusted channel.

To prevent this situation in this thesis, an additional annotation is attached to the deduction itself, indicating the lowest integrity level of channels that it relies on. In the example above this is clearly untrusted, so little guarantees can be made about its security (as the example demonstrates).

1.2.2 Higher-Order π-calculus

There is no support in the plain π-calculus for transmission of processes, as might be used to represent mobile code. However, similar effects can faithfully be achieved through link-activation: that is, instead of sending the process itself, a private link to the process is sent which may then be used to synchonise in order to "run" it.

For example, if processes are allowed as channel data, then the scenario of P sending process R to be executed by Q might be represented as follows:

$$\overline{x}[\mathrm{R}].\mathrm{P} \mid x(X).(X|\mathrm{Q}) \rightarrow \mathrm{P} \mid \mathrm{R} \mid \mathrm{Q}$$

(where $\rightarrow$ denotes reduction). An alternative encoding that only transmits channel access is just a little more verbose:

$$\begin{array}{rcl} \overline{x}[y].\mathrm{P} \mid y.\mathrm{R} \mid x(z).\overline{z}.\mathrm{Q} & \rightarrow & \mathrm{P} \mid y.\mathrm{R} \mid \overline{y}.\mathrm{Q} \\ & \rightarrow & \mathrm{P} \mid \mathrm{R} \mid \mathrm{Q} \end{array}$$

Note that there is an extra reduction step, but the end result is the same (a factor in this is that the language has no concept of locality). In this scenario it is obviously desirable that the name y be private until transmitted, and the π-calculus provides facilities for local name creation. These will be covered in Chapter 2.

It has in fact been demonstrated that such a higher-order version of the language may be faithfully encoded (Sangiorgi 1993a) in the first-order system considered up to now, but the extra expressivity it provides make it an attractive meta-language in its own right.

Of course, given that processes may be transmitted on channels, and some channels may expose their contents to the risk of corruption, it seems a natural consequence that any type system in a similar vein for a higher-order π-calculus must also contain a notion of process integrity, as well as channel integrity. This in turn implies that just as multiplication of annotations expresses that the contents must be untrusted if the channel is, a similar

condition must be imposed so that data coming from an untrusted source is also untrusted (even if the channel is secure).

This situation is more complicated that it seems though: when two processes are running in parallel, each with their own integrity level, how should they be combined? Multiplication of annotations (as used above to express dependencies between channels and their contents) in a boolean algebra produces the lower bound, so if one process is considered untrusted then so is any parallel combination including that process. If, however, the untrusted process is incapable for whatever reason of actually inflicting any harm then this would seem to be over-kill.

A little thought along these lines reveals that a more flexible viewpoint is "it depends". In particular it should depend on a process' ability, particularly that of an untrusted process, to cause harm. This ability must necessarily also be dependent on the host running the processes; consider the difference between a malicious applet running in a sandbox with no access to the filesystem, network, etc, versus the same applet running in an unfettered environment. Clearly, the former may be considered trusted and allowed to execute, while the latter should definitely be considered untrusted.

There are two components to the implementation of this perspective in this thesis:

- A deduction captures the *context* that every name is used under. This is a mapping between the free names of the deduction, and the trustedness of the process that uses them (not the trustedness of the name itself). So if an untrusted process performs an input on the channel x, x would be in an untrusted context in that deduction, irrespective of the type of x.

- Secondly, every deduction is performed relative to a security policy, termed the external context (represented as $\mathbb{C}_\epsilon$). This represents the host's open ports of access available to the program.

The final step in a deduction is then to combine the contexts of all names that the host exposes (all those in the external context) to calculate the possibility of a program harming the host.

An example should help to make these ideas clearer. Assume the following malicious (untrusted) process $\overline{\mathit{filesystem}}[\mathit{junk}].\mathrm{P}$, where *filesystem* is the channel used to read and — in this case — write to the file system. This simple example represents a malicious process intent on writing junk to the host file system. Given that P has previously been deduced to be untrusted, then the channel *filesystem* appears in an untrusted context in this instance.

The next step is to examine the contents of the external context. If the host allows access to the file system, the external context will contain *filesystem*, and thus the final deduction should be that the program is untrusted. However, if the host runs all programs in a sandbox, with no access to the file system, then despite the working assumption of untrustedness it is safe for the host to run *with this security policy* (that is, the final deduction will be that the process can be trusted). In the case of multiple names from the security policy appearing in the context they are combined by taking the conjunction.

1.3 Coercion Based on Verification

An interesting approach that blends formal static analysis with respect for the programmer's knowledge is one which allows the programmer to guide the analysis by marking parts of the code as safe (or alternatively unsafe). The most oft-cited example of this is the work of Ørbæk and Palsberg (Ørbæk 1995; Palsberg and Ørbæk 1995), who presented systems of both a functional and imperative nature along these lines.

Their approach is best illustrated by an example:

Example 1.3.1 *The following example is from Palsberg and Ørbæk (1995, Section 1.2); it represents a simple server processing network requests with an accompanying digital signature. The server code (using a SML-style syntax) is:*

```
fun getRequest client =
  let (req, signature) = readFromNetwork(client) in
    if verifySignature(signature) then
      handleEvent(trust req)
    else
      handleWrongSignature(req, signature)
```

where the handler code resembles:

```
fun handleEvent req =
  let trustedReq = check req
  in ...
```

The two pertinent syntactical additions are the phrase `trust req` *in the server, and* `check req` *in the handler code. The intention is that the request* `req` *is initially untrusted (and thus will be detected by the static analysis if improperly used), but if some verification procedure is successful — in this*

case checking the signature — the programmer coerces *the data to trusted using the* `trust` *keyword. Where it is essential that data be trusted, the keyword* `check` *is used, which only type-checks if its argument can be* statically *determined to be trusted.*

This simple combination of three operators (the `distrust` *keyword is also available to perform the opposite function to* `trust` *if necessary) turns out to be very powerful; the authors claim that only a minimal usage of the extra operators is typically required, and it provides all the benefits of a static analysis with the additional flexibility of allowing the programmer to override (or gently guide) it when necessary.*

This is quite a pragmatic approach, and probably appealing to programmers frustrated by overly-conservative analyses. While much-cited however, any further analysis of such systems is rather scant. The reason for this is likely to be the same reason just quoted as a benefit: while it allows the programmer to guide the analysis where required, it implicitly puts *too* much control in the hands of the programmer, the dangers of which can be seen in Example 1.3.2:

Example 1.3.2 *Revisiting Example 1.3.1, omitting the handler code (which remains the same), consider what happens if the* `verifySignature` *branches in the server code get mixed up; that is:*

```
...
  if verifySignature(signature) then
    handleWrongSignature(req, signature)
  else
    handleEvent(trust req)
```

The issues in this example are easily apparent: a simple programmer error results in code that still type-checks and would be deemed safe according to the additional annotations, but is obviously seriously flawed — the integrity check may fail, but the request is still marked as trusted. The primary concern is that `trust` is an *explicit* cast, and its correct usage is not (and cannot be) checked by the analysis.[1] In other words, its results are only useful modulo correct usage of its facilities by the programmer, which most people would regard as still too dangerous. Nonetheless, it is a most promising direction, and will be investigated further in this thesis.

[1] This also assumes that the programmer is *trying* to do the right thing; picture a junior programmer faced with deadlines, inserting `trust` in numerous places to fix the safety error the compiler keeps reporting!

1.3.1 An Alternative Approach

This thesis suggests an alternative approach that encompasses the pragmatic benefits of programmer-driven coercion, but in a safe manner by making the coercion itself *implicit*. It is contended that the structure of the code in Examples 1.3.1 and 1.3.2 is relatively common: perform some form of verification procedure, and branch based on the results (typically to the main code if verification is successful or to error-handling code if it fails, as in the example). The issues with the `trust`/`check` keywords are that they are disconnected from each other, and importantly they are not related in any way — `trust` in particular — to the verification procedure used (if there even is one).

The solution proposed in this thesis is an addition to the language of a single construct — called *certify* — that encompasses *all* aspects: verification, branching, and coercion. This instantly removes the issues raised earlier, whilst retaining the benefits:

- coercion is available to the programmer, but only dependent on some form of certification;
- the branching performed is now also *directly* related to the certification procedure; and
- the coercion is implicit in the branch chosen; for example the first branch is always parameterised on a trusted variable, the second on an untrusted variable: if the branches are confused, the static analyser (the type system) will now detect it.

1.3.2 Implementation

There are two factors to consider when implementing this new idea in the π-calculus: how it is incorporated into the language, and the implications for the type system.

To begin with, as stated, the implementation of the verification procedure itself are left abstract. The existence of an oracle is assumed, ranged over by certify, represented as a function. That is, it is assumed that certify(x) correctly deduces the integrity of the channel x at run-time.

This is then incorporated into the language with a construct modelled on the input syntax, but with two separate bodies:

$$x(y)^{?}_{\text{certify}}\mathrm{P} \oplus \mathrm{Q}$$

This performs similarly to an input process, receiving a value on the channel x, and binding the value to y. The difference is that the oracle certify is

first used to determine its trustedness; if it is found to be trusted it is bound to y in P, and if untrusted to y in Q. The other branch is discarded. In this manner it combines the complementary operations of verification and branching.

Secondly, the incorporation of this new operation into the type system must be considered. There are several elements that must be combined. Firstly, in the typing of the certified input construct above, the bound input name y has a slightly different type in each branch, being trusted in P and untrusted in Q. This lets the programmer access coercion and verification, but without the ability to abuse it indiscriminately.

In order to perform coercion, variable annotations must be admitted into the annotation algebra. This has additional expressive benefits: a channel or value with a variable annotation represents a trustedness that cannot be determined at compile-time but *can* be determined at run-time, for example via byte-code or signature verification. At run-time when verification is performed this variable is coerced to a concrete (trusted or untrusted) value globally.

This expressiveness can also be applied in other areas, such as in calculating the combined security level of a certified input construct. Assuming that annotations are ranged over by b, c and annotations variables by i, then supposing that process P is typed at level b and Q at level c, where the input to be verified has a run-time trustedness of i, the combined level can be elegantly expressed as $i \cdot b + \bar{i} \cdot c$. In other words, if i is mapped to trusted this expression will reduce to b, which is the level of P, the branch that is executed. Similarly it will evaluate to the level of Q if i is mapped to untrusted.

Expressing the semantics of this operator is slightly tricky, and is here accomplished using a typed form of labelled transition semantics. Plain labelled transition semantics are a common way of describing the semantics of process algebra; rules are expressed in the form $\mathrm{P} \xrightarrow{\mu} \mathrm{P}'$, to describe that process P may perform action μ and reduce to P' in the process. This must be elaborated on to describe the certify operator. A rule of the form

$$\langle \Theta, \Gamma, \mathrm{P} \rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}' \rangle \Vdash_{\text{certify}} \mathbb{R}$$

states that process P, when typed under environment Γ, may possibly interact with an anonymous second process typed under environment Θ, to perform the action μ and transform itself to P' and the two environments to Γ' and Θ' respectively. Additionally, the reduction generates the function $\mathbb{R}$ which is a substitution from annotation variables to annotations, and which represents every coercion that occurs due to certify in the reduction. While

more complicated, this allows a precise specification of the names that are exchanged between processes, and crucially the coercions that occur on the type annotations.

1.4 Related Work

This section summarises the existing work in related areas, and makes comparisons with the approach of this thesis where warranted; additional comparisons are presented at the end of Chapter 3, when they can be discussed in context.

As noted by Dorothy Denning (Denning 1999) it is impossible to formally *prove* any system is secure as by definition this implies secure *with respect to a given model*, and attacks typically arise around the model (for instance, using social engineering to obtain a password, to choose a blunt but salient example).

None-the-less, this does not detract from the value of formal analysis, merely suggests that it should be considered as part of a broader context.

It is convenient when examining related approaches to group them roughly by the topics studied in this thesis. These are, coarsely: information flow analysis, the use of coercion in integrity or flow analysis, and the analysis of mobile code (higher-order systems).

1.4.1 Information Flow Analysis

Information flow analysis is the process of protecting the secrecy levels of information; it is probably also one of the earliest formal security analyses (Denning 1976).

Such analyses typically assume a range of secrecy levels, often expressed as a lattice, and seek to assure the user that a program will not leak classified information to a member (for example; variable) with a lower classification. The reverse flow, from less secret to more secret, is typically allowed. Security properties are usually expressed as some form of observability, typically that an observer at some security level can only observe data at an equivalent or lower classification. Some examples will help summarise the principles involved.

Example 1.4.1 *Consider the following (imperative) program statement:*

```
x := y
```

In this case there is a clear flow of information from `y` *to* `x` *by direct assignment, hence this statement can only be approved if* `x` *has a* higher *security*

clearance than `y`*. The situation is potentially more complicated however, for example the following fragment:*

```
if (x < 5) then y := 0 else y := 3
```

Here there is no explicit assignment from `x` *to* `y`*, but a user with access to* `y` *can* still *glean some information about* `x`*: specifically if its value is less than five or not, and thus there is an* implicit *flow from* `x` *to* `y` *which must also be monitored. A similar situation naturally exists for while–loops and other related constructs.*

In most respects secrecy analysis is in fact the dual of integrity analysis: secrecy analysis aims to ensure that no data may flow from a higher security classification to a lower, whilst integrity analysis seeks to ensure that low integrity data is not written to a high-integrity variable. In other words, a high integrity level corresponds to a low secrecy level, and vice versa (this observation is in fact first due to Biba (1977)).

The formal analysis of secrecy flow began with Dorothy Denning (Denning 1976). This research used a lattice model of security levels, and provided a framework for their implementation and analysis.

This direction was later extended and implemented as a *type system* for a sequential imperative language in Volpano et al. (1996). Formalisation as a type system enabled a demonstration of soundness of the model, in the form of a non-interference result. These results were extended further still to examine a multi-threaded imperative language in Smith and Volpano (1998), where the issue of non-interference is complicated by the possibility of timing attacks and the like (interestingly, their results depend on the nature of the thread-scheduling algorithm: a non-deterministic algorithm is preferred, and a more restrictive type system is required if a deterministic algorithm is deployed).

Recent results in the area are plentiful.

Pottier and Conchon (2000) describe a method of systematically extending a functional type system to derive a system of information flow. Bodei, Degano, Nielson and Nielson (2001) present an analysis for the π-calculus capable of determining the subset of channels a name may be bound to at runtime, and derive several common security properties. By necessity though this loses alpha-convertibility and requires a minor syntactical adjustment, although it is unclear if this is really an impediment or not.

Pottier (2001) provides a relatively simple method of reducing a proof of non-interference to subject reduction, for a non-standard extension of the π-calculus.

Abadi, Banerjee, Heintze and Riecke (1999) reduce several different analyses (security flow, binding-time, slicing, call-tracking) to a common core in their Dependency Core Calculus. Thus multiple analyses may be derived by encoding the parent calculus in the core calculus, and using the results established for the core calculus. They also offer an encoding of the purely-functional fragment of the SLam calculus (although as with most results derived from Palsberg and Ørbæk (1995), they do not encode coercion from untrusted to trusted).

The first-order π-calculus is also used in both Honda et al. (2000) and Honda and Yoshida (2002), with a relatively complicated type system annotated with both linear/affine input/output properties (in the manner of Kobayashi, Pierce and Turner (1996)) and secrecy levels. A sound encoding of the multi-threaded imperative calculus of Smith and Volpano (1998) is proffered for both; the latter (Honda and Yoshida 2002) is powerful enough to encode the Dependency Core Calculus (Abadi et al. 1999).

Myers and Liskov (1997) analyse a decentralised system that offers fine-grained control over data-sharing by allowing the users to specify data-flow policies as program annotations. In addition, they provided a statement `declassify(e,L)` that allowed the user to explicitly control downgrading of data classification, although the safety of this operator is not established in this work (this topic will be examined in more detail in Section 1.4.2). Myers (1999) provided a practical implementation as an extension of the Java language (Gosling, Joy, Steele and Bracha 2000), although some features were altered or missing due to unsolved research questions (threads in particular). This work is on-going at *Jif: Java + information flow* (2005).

A different approach is taken in Abadi and Gordon (1997) and Abadi (1997), where a version of the π-calculus enriched with primitives for describing encryption is presented, enabling reasoning about the safety of cryptographic protocols by type-checking (rather than, say, theorem-proving). The security property is an observability result; a well-typed protocol leaks no information to an outside observer. An implementation of related ideas exists in the cryptyc tool (Gordon and Jeffrey 2001; *Cryptyc: Cryptographic Protocol Type Checker* 2006).

1.4.2 Approaches to Coercion

The SLam calculus (Secure Lambda) (Heintze and Riecke 1998) is a variant of the lambda calculus that maintains security as well as type information. It is focused on secrecy preservation (as opposed to integrity protection) and is thus essentially a form of flow analysis (see Section 1.4.1), however it also offers a limited form of coercion to the programmer via the `protect`

construct. This is, however, a one-way coercion and can only be used to make the security level of its argument *more* restrictive (and thus avoids the pitfalls outlined above).

The ideas in Myers and Liskov (1997) were refined in Zdancewic and Myers (2001). This work formalised the extent to which information is leaked using their `declassify` operator by defining a notion of *robust declassification*. The analysis is based on an equivalence relation over a state-transition system, and argues the point that many useful systems *must* leak information as part of their function (the example of a password-checking system was used; the password must be kept private, but a failed login leaks a small amount of information about what the password is not). No mechanism for enforcing a safe policy involving intentional downgrading of information was offered; this was presented in Myers, Sabelfeld and Zdancewic (2004). This work provided a more language-based presentation (for an imperative language), and offered a refined notion of robustness called *qualified robust declassification*, which allows an attacker limited ability to control information release. A type system was presented, such that well-typed programs satisfied the robustness condition.

This work is interesting from a second perspective; their implementation used a lattice incorporating both confidentiality *and* integrity. Two coercion-like operators were used; `declassify` which lowers the confidentiality classification for a fixed integrity level, and *endorse* which increases the integrity classification without affecting the confidentiality. The interaction between these two operators, which is quite delicate as `declassify` requires a fixed integrity level and `endorse` alters integrity levels, is also carefully studied.

1.4.3 Higher-Order Systems and Mobile Code

While security analyses utilising the first-order π-calculus or some variant abound (most a form of flow analysis, see above), there are comparatively few dealing with the more difficult angle of mobile code, or higher-order systems.

The work of Myers (1999) has been extended in Zdancewic, Zheng, Nystrom and Myers (2001) to automatically partition a program to run in a distributed system. A prototype implementation as an extension of Java also exists.

Confined-λ (Kirli 2001) is an elegant combination of the λ-calculus extended with operators for sending and receiving data along channels (much like the π-calculus). Since functions are also data, it is a true higher-order system.

Their approach is slightly different to most examined in this chapter;

rather than providing an ordering of security classifications, the concept of *mobility regions* is used. These are essentially sets of users (or computation sites) allowed to view each function. While they suggest that this is more simplistic than the richer structures of the SLam calculus (with which a comparison is made), it would appear to be a practical way of enforcing secrecy in distributed systems (by restricting a computation to only known sites). Their security property states that well-typed programs restrict values to their intended region. It is also suggested that it may approximate more refined notions of secrecy discussed elsewhere by using mobility regions to represent the differing security classes (since regions may be nested in other regions).

Yoshida and Hennessy (2000) offer a more fine-grained approach. The modelling language is the higher-order π-calculus, and their type system borrows from the λ-calculus. Processes are typed with an *interface*, where an interface associates names with capabilities (input and output usage, and the types of those names). This enables other processes to decide whether or not to run processes based on their interface, for example. An interesting extension of this idea is also included, using a kind of *dependent type*: these are used to extend an interface to express process abstractions, for example $(x : \sigma) \rightarrow \rho$, where ρ is an interface and x a channel variable with type σ (ρ is allowed to contain occurrences of x).

An extension to that work is presented in Vivas and Yoshida (2002) where a dynamic operator is used to preserve many of the properties that were statically checked in Yoshida and Hennessy (2000). Their new operator is a screening operator that provides a more powerful notion of restriction than the regular π-calculus restriction operator (that is, $(\nu x)\mathrm{P}$), which loses encapsulation due to scope extrusion. The most interesting thing about this work (from this perspective) is the dynamic nature of their operator, although it is in other ways analogous to the external context ($\mathbb{C}_\epsilon$) of this system. For example, the reduction

$$\mathrm{P}\lceil L \xrightarrow{\mu} \mathrm{P}'\lceil L$$

is only defined if $\|\mu\| \in L$ where L is the filter, a set of names denoting allowable actions (for additional granularity, names are also marked by a directional tag; stating for instance that x may only be used for input across the filter, see Section 6.1.1 for additional details). A proof is provided that this adequately encodes the system in Yoshida and Hennessy (2000). Focardi and Gorrieri (1995) used a similar operator in their Security Process Algebra (a metalanguage based on CCS) to classify and compare a range of security properties for non-deterministic systems.

Lastly, it is worth mentioning a most interesting approach to ensuring

safety of mobile code: proof-carrying code (Necula 1998; Necula and Lee 1998). This is a novel approach in which untrusted code is only allowed to run if it can prove that it respects certain security properties set by the host. The security requirements are stated as logical axioms, and the proof of a program encoded as a binary representation of the proof. The size of the proof reportedly outweighs that of the program by several factors, and the proof itself is not easy to construct: the interesting thing however is that the burden of proof is on the program author, not (as is commonly the case in most systems examined here) on the host. It is a relatively quick matter for the host to verify both that the proof is correct and that the program respects the proof, and if so it may execute the program.

This is clearly unlike most other analyses presented here and has no analogue in this thesis; however it is of interest in that it provides one method by which certify may verify a piece of code as being safe.

1.5 Dissertation Structure

The remainder of the dissertation is structured along fairly standard lines. While at least a passing familiarity with the π-calculus would be advantageous, no previous knowledge is assumed and Chapter 2 provides a summary overview, including type systems and syntax for both first- and higher-order versions of the calculus.

The novel work begins in Chapter 3, with a description of the type systems and a syntactic addition that forms the core of the thesis. Four systems are presented; three progressively more complicated first-order systems, and a higher-order system. *Safety* of the systems follows in Chapter 4, and is presented as a statement of subject reduction for each system, and a security property in the form of a strong non-deterministic non-interference result for the first-order system, and an observability-based property for the higher-order system. Only three of the four systems are examined from this perspective, as one is mostly a vehicle to develop a more sophisticated system. Chapter 5 describes implementations for each system in the form of type inference algorithms and proves their correctness, and finally Chapter 6 concludes and suggests some directions for future work.

Chapter 2

Preliminaries

2.1 Introduction

In this chapter a formal treatment of the topics that form the basis for the thesis is presented, in particular the π-calculus and the foundations of its type system.

The calculus is first introduced via its syntax and semantics, of both the first and higher-order calculi. The operational semantics are presented in terms of a reduction relation and also as a labelled transition system.

The type systems are presented separately from the syntax, and both first and higher-order systems are examined. Sub-typing is not covered: sub-typing for the π-calculus typically involves some extra information appended to the basic type (usually including a notion of input/output usage; see Pierce and Sangiorgi (1993) for the first work in this area) and this chapter restricts itself to the fundamental type systems. Sub-typing will be covered in the context of the novel systems presented here, beginning in Chapter 3.

2.2 The π-calculus

The π-calculus was originally proposed by Milner, Parrow and Walker (1992a,b) based on Milner's early work on CCS (Milner 1980), as a way of reasoning about concurrency in a similar way that the λ-calculus enables reasoning about computation. In keeping with the minimalist nature of the λ-calculus, the core π-calculus is very small: the two fundamental concepts are parallel execution (or concurrency), and communication. Processes may execute in parallel with one-another, and reduce by communication of data along named channels. In its purest form, all data is made up of channels, and processes themselves are constructed simply from channels. This simple core provides

a lot of flexibility: new links can be established by sending the corresponding channel to a process that previously did not possess it. It has previously been demonstrated (Milner et al. 1992a) that the π-calculus can encode the λ-calculus (although note that one must also decide on the reduction scheme at compilation time), so in theory the π-calculus offers all the benefits of the λ-calculus, together with the power of concurrency.

This thesis will use a variant of the polyadic π-calculus as described in Milner (1993) (*polyadic* refers to the ability to send multiple values along a channel in a single transmission. The original π-calculus presentations were *monadic*; that is, only a single value could be transmitted at a time). There is no formal reason for this choice, but it would appear to offer greater flexibility than the monadic calculus and is in keeping with most current research; and in any case the results extend downwards to the monadic calculus.

Some examples of the calculus in use will be provided first, followed by a formal presentation of the syntax.

Firstly, consider the case of a process denoted by P, executing in parallel with a second process denoted by Q: in the π-calculus this situation is concisely denoted by P|Q.

Secondly, consider an example of communication. For two processes to communicate it is necessary that they have the same name (channel) at the outermost level (this situation is analogous to a socket in common network programming APIs). A name used in an output context is written with a bar over the top, and the names that are being sent are enclosed in square brackets immediately afterwards. Thus a process that outputs the single name y along a channel x, then continues as the process P would be written as $\overline{x}[y]$.P. The corresponding input case is written similarly: the channel being used for input is written followed by the bound variable (formal parameter) in parentheses; so a process that receives a single name along a channel x and binds it to a variable z in a process Q is written as $x(z)$.Q. (Note that in the version of the π-calculus considered here it is possible to communicate tuples of names together, although this is not demonstrated in the examples just given).

Now a brief but instructive example will be examined to demonstrate the modelling capabilities of these concepts.

Example 2.2.1 *Consider the example of a file-sharing program, in which a central server maintains a list of files stored on each node. When a particular client wishes to download a file (that it has presumably discovered on the centrally-stored list), it notifies the server which responds with the* address *of the node storing that particular file. The requesting client may now establish a link with the host node and download the file.*

This may be described in the π-calculus as follows. Let the server process be represented by server*, the client by* client*, and the host by* host*. Then a representation of the above scenario is:*

$$\overline{s}[\text{req}].\text{req}(h).h(\text{disk}).\text{client} \mid s(c).\overline{c}[\text{link}].\text{server} \mid \overline{\text{link}}[\text{file}].\text{host}$$

The client begins by connecting to the server (along channel s) and sending its request req*, which the server binds to the channel c, producing the following:*

$$\text{req}(h).h(\text{disk}).\text{client} \mid \overline{\text{req}}[\text{link}].\text{server} \mid \overline{\text{link}}[\text{file}].\text{host}$$

The server then uses the req *channel provided by the client to send the address — called* link *— of the host possessing the file; leading to:*

$$\text{link}(\text{disk}).\text{client} \mid \text{server} \mid \overline{\text{link}}[\text{file}].\text{host}$$

at which point the client may start downloading file *from* host *(and writing it to its internal variable* disk*).*

This is a generalisation of a relatively common scenario in peer-to-peer networks; the main principle though is the expressive power of the π-calculus. To begin with, the only process with knowledge of another was the client (that was aware of the server), and the server that was aware of the host. Yet, by passing around addresses, the network is able to dynamically reconfigure itself by creating new links.

2.2.1 First-order π-calculus

The original and most common versions of the π-calculus are *first order*; that is, only channels can be transmitted as data and not whole processes. This may seem overly restrictive, but perhaps surprisingly it is a powerful enough abstraction that the λ-calculus can be encoded in it (Milner et al. 1992a). The key to this power is the dynamic topologies of the networks created in this manner; because new links can be established on the fly, processes can in effect reconfigure themselves dynamically to provide much the same effect as transmitting whole processes (see also Sections 2.2.2 and 2.2.3 for further discussion of this matter).

Syntax

The syntax adopted here is in line with most modern presentations; it is generally simpler than that presented in Milner (1993). This is for reasons of clarity; the concise syntax used here is sufficient, and most proofs encountered are by structural induction and benefit from the smaller number of cases.

The first-order π-calculus is given by the grammar of Definition 2.2.2:

Definition 2.2.2 (Terms) *Let processes be ranged over by* P, Q, *and* R, *and the set of all processes be represented by* $\mathcal{P}$. *Let names be ranged over by* x, y, *and* z, *and the set of all names be represented by* $\mathcal{N}$ *(the terms* name, port, *and* channel *will be used interchangeably, with a preference for "name").*

Terms of the first-order π*-calculus are defined inductively as the least set containing*

$$\begin{aligned} \mathrm{P} &::= \mathbf{0} \mid \sum_{i \in I} \pi_i.\mathrm{P}_i \mid \mathrm{P}|\mathrm{P} \mid !\mathrm{P} \mid (\nu x)\mathrm{P} \\ \pi &::= \overline{x}[\overrightarrow{y}] \mid x(\overrightarrow{y}) \end{aligned}$$

Note that summation is only permitted for guarded *processes; that is, those with an outer-most input or output prefix (Milner 1993). Processes of the form* $(\mathrm{P}|\mathrm{Q}) + \mathrm{R}$ *are not permitted. This is mainly to simplify the semantics and some of the proofs that will follow. As a shorthand notation the indexing set* I *in the summation term will often be omitted, and such processes will usually be written as* $\mathrm{P} + \mathrm{Q}$ *anyway.*

Following convention, terms such as $\overline{x}[\overrightarrow{y}].\mathbf{0}$ *will often be abbreviated to* $\overline{x}[\overrightarrow{y}]$ *(that is, where it is clear the null process "base" may be omitted).*

Informally (a more rigorous treatment follows); $\mathbf{0}$ is the inactive process, $x(\overrightarrow{y}).\mathrm{P}$ inputs names along a channel named x and binds them to $\overrightarrow{y}$, $(\nu x)\mathrm{P}$ creates a new local name x (and thus binds x), and $\overline{x}[\overrightarrow{y}].\mathrm{P}$ is the (non-binding) output construct. Summation represents a discriminated choice; $\mathrm{P} + \mathrm{Q}$ may reduce as *either* P or Q, but not both. Summation is also a *non-deterministic* choice, although in practice the decision of which executes is determined by whichever has prior opportunity (for example, summation is often used when encoding booleans and an "*if . . . then . . . else*" programming construct). Parallel execution is denoted by $\mathrm{P}|\mathrm{Q}$, while replication ($!\mathrm{P}$) describes an infinite supply of process P; it may be defined as $!\mathrm{P} \triangleq \mathrm{P}|!\mathrm{P}$, where the reduction rules below will make sure it does not simply replicate continuously.

In all the work that follows it is assumed that all bound names have been renamed to be distinct (see Convention 2.2.4). The only two binding constructs are input, for example $x(\overrightarrow{y}).\mathrm{P}$, which binds $\overrightarrow{y}$; and restriction, for example $(\nu x)\mathrm{P}$, binding x in P. The bound and free names are defined in the usual way (Definition 2.2.3):

Definition 2.2.3 *Let the free names of a term* P, *written as* $FN(\mathrm{P})$ *be de-*

fined inductively as follows:

$$\begin{array}{rcl@{\qquad}rcl} FN(\mathbf{0}) & \triangleq & \emptyset & FN(\mathrm{P}|\mathrm{Q}) & \triangleq & FN(\mathrm{P}) \cup FN(\mathrm{Q}) \\ FN(!\mathrm{P}) & \triangleq & FN(\mathrm{P}) & FN(x(\overrightarrow{y}).\mathrm{P}) & \triangleq & \{x\} \cup FN(\mathrm{P}) - \{\overrightarrow{y}\} \\ FN((\nu x)\mathrm{P}) & \triangleq & FN(P) - \{x\} & FN(\overline{x}[\overrightarrow{y}].\mathrm{P}) & \triangleq & \{x, \overrightarrow{y}\} \cup FN(\mathrm{P}) \\ & & & FN(\Sigma_i \mathrm{P}_i) & \triangleq & \bigcup FN(\mathrm{P}_i) \end{array}$$

The bound names of a term (written $BN(\mathrm{P})$) are defined similarly.

Following Barendregt (Barendregt 1981, Variable Convention 2.1.13), all bound names are renamed to be distinct (Convention 2.2.4). Note that this is consistent with the Definition (2.2.5) of structural congruence in which terms are identified up to alpha-convertibility.

Convention 2.2.4 *If $\mathrm{P}_1, \ldots, \mathrm{P}_n$ occur in a certain mathematical context (for example, definition, proof), then in these terms all bound names are chosen to be different from the free names.*

Structural congruence over processes is defined in Definition 2.2.5:

Definition 2.2.5 *Structural congruence, written $\equiv$, is the least transitive reflexive relation satisfying the following:*

- $\mathrm{P} \equiv \mathrm{Q}$ *if* P *is α-convertible to* Q *(that is, if they are identical up to renaming of bound names)*
- $!\mathrm{P} \equiv \mathrm{P} \mid !\mathrm{P}$
- $(\nu x)\mathbf{0} \equiv \mathbf{0}$; $(\nu x)(\nu y)\mathrm{P} \equiv (\nu y)(\nu x)\mathrm{P}$; $(\nu x)\mathrm{P}|\mathrm{Q} \equiv (\nu x)(\mathrm{P}|\mathrm{Q})$
- $\mathrm{P}|\mathrm{Q} \equiv \mathrm{Q}|\mathrm{P}$; $(\mathrm{P}|\mathrm{Q})|\mathrm{R} \equiv \mathrm{P}|(\mathrm{Q}|\mathrm{R})$; $\mathrm{P}|\mathbf{0} \equiv \mathrm{P}$ *(that is, Abelian monoid laws based around $|$, with $\mathbf{0}$ as the identity element)*
- $\mathrm{P}+\mathrm{Q} \equiv \mathrm{Q}+\mathrm{P}$; $(\mathrm{P}+\mathrm{Q})+\mathrm{R} \equiv \mathrm{P}+(\mathrm{Q}+\mathrm{R})$; $\mathrm{P}+\mathbf{0} \equiv \mathrm{P}$ *(that is, Abelian monoid laws based around $+$, with $\mathbf{0}$ as the identity element)*

If P *is alpha-convertible to* Q *this will sometimes be written as $\mathrm{P} \equiv_\alpha \mathrm{Q}$, where $\equiv_\alpha$ is the least reflexive, transitive, and symmetric relation defined when its operands are alpha-convertible to each other.*

Name substitution on first-order terms is specified in Definition 2.2.6:

Definition 2.2.6 (First-Order Substitutions) *For a given term* P*, write $\mathrm{P}\{y/x\}$ to denote the same term with all free occurrences of x replaced by y. Let $\overrightarrow{x}$ refer to a sequence of names $x_1 \ldots x_n$ and $\overrightarrow{y}$ the (distinct) sequence*

$y_1 \ldots y_n$ (for some finite n); then the definition may be extended to cover sequences of names, for example $\mathrm{P}\{\overrightarrow{y}/\overrightarrow{x}\}$ *(note that by definition the lengths of both sequences are identical, and all* $x \in \overrightarrow{x}$ *are assumed to be distinct).*

More formally, this can be expressed inductively:

$$\begin{aligned}
\mathbf{0}\{y/x\} &\triangleq \mathbf{0} \\
x\{y/x\} &\triangleq y \\
z\{y/x\} &\triangleq z \quad (z \neq x) \\
!\mathrm{P}\{y/x\} &\triangleq !(\mathrm{P}\{y/x\}) \\
(\mathrm{P}|\mathrm{Q})\{y/x\} &\triangleq (\mathrm{P}\{y/x\})|(\mathrm{Q}\{y/x\}) \\
(\Sigma_i \mathrm{P}_i)\{y/x\} &\triangleq \Sigma_i(\mathrm{P}_i\{y/x\}) \\
(\nu z)\mathrm{P}\{y/x\} &\triangleq (\nu z)(\mathrm{P}\{y/x\}) \\
(z(\overrightarrow{z'}).\mathrm{P})\{y/x\} &\triangleq z\{y/x\}(\overrightarrow{z'}).(\mathrm{P}\{y/x\}) \\
(\overline{z}[\overrightarrow{z'}].\mathrm{P})\{y/x\} &\triangleq \overline{z\{y/x\}}[\overrightarrow{z'}\{y/x\}].(\mathrm{P}\{y/x\})
\end{aligned}$$

(In the above, the convention is used that if $\overrightarrow{z}$ *represents the sequence* $z_1 \ldots z_n$ *then* $\overrightarrow{z}\{y/x\}$ *represents* $z_1\{y/x\} \ldots z_n\{y/x\}$*).*

Reduction Semantics

Reduction semantics are presented as *term-rewriting* rules; this enables describing the operation of the language by showing a complete reduction in the context in which it occurs. The key to describing a reduction semantics in the π-calculus is structural congruence (see Definition 2.2.5): in the λ-calculus redexes appear contiguously as sub-terms, whereas in a process calculus describing mobility (such as the π-calculus) by definition this is not necessarily the case. The addition of a congruence rule mitigates this situation by enabling manipulation of terms so that redexes do in fact appear as sub-terms for the purposes of presentation (Milner 1993).

There is only a single reduction *axiom*; one describing communication of names along named channels. There are three structural rules defining the circumstances under which reduction can (and, just as importantly, cannot) occur.

Communication reduction occurs when one process sends some (possibly zero) names along a channel, where they are bound to names in the receiving process (note how this rule also expresses the exclusivity of summation):

$$\overline{(\ldots + \overline{x}[\overrightarrow{y}].\mathrm{P}|x(\overrightarrow{z}).\mathrm{Q} + \ldots) \rightarrow \mathrm{P}|\mathrm{Q}\{\overrightarrow{y}/\overrightarrow{z}\}}$$

To this axiom are then added three inference rules; stating respectively

- reduction may occur under a restriction (but *not* under a prefix):

$$\frac{\mathrm{P} \to \mathrm{P}'}{(\nu x)\mathrm{P} \to (\nu x)\mathrm{P}'}$$

- structurally congruent terms have the same reductions:

$$\frac{\mathrm{P} \equiv \mathrm{Q} \quad \mathrm{Q} \to \mathrm{Q}' \quad \mathrm{Q}' \equiv \mathrm{P}'}{\mathrm{P} \to \mathrm{P}'}$$

- reduction may occur in parallel:

$$\frac{\mathrm{P} \to \mathrm{P}' \quad \mathrm{Q} \to \mathrm{Q}'}{\mathrm{P}|\mathrm{Q} \to \mathrm{P}'|\mathrm{Q}'}$$

The Kleene-closure (that is, the reflexive and transitive closure) of the arrow $\to$ is written as $\twoheadrightarrow$.

It is also necessary to ensure that names under a restriction can be emitted outside the restriction where necessary and if it doesn't cause any confusion. This is known as *scope extrusion* (Definition 2.2.7):

Definition 2.2.7 (Scope Extrusion) *The scope of a restriction binder may be extended as far as necessary, providing it doesn't capture other names in doing so. In particular, the scope may be extended to enable names under the scope of a restriction to be emitted to processes formerly outside the scope.*

For example, assuming that $y \notin BN(\mathrm{Q})$*:*

$$(\nu y)\overline{x}[y].P \mid x(z).Q \to (\nu y)(\mathrm{P} \mid \mathrm{Q}\{y/z\})$$

Note that scope extrusion is achieved in the reduction semantics via a clause in the definition of structural congruence (Definition 2.2.5). Using the previous example, the reduction would proceed as

$$\begin{aligned}(\nu y)\overline{x}[y].P \mid x(z).Q &\equiv (\nu y)(\overline{x}[y].P \mid x(z).Q) \\ &\to (\nu y)(\mathrm{P} \mid \mathrm{Q}\{y/z\})\end{aligned}$$

Labelled Transition Semantics

If the benefits of a reduction semantics are its naturalness, then the advantages of a labelled transition system lie in its ability to express the possible communications of a sub-term independent of its context. This will prove especially useful when *reasoning* about processes (in particular, concerning security properties), as it enables a precise specification of the property and

its proof — avoiding confusion about the role of the context. The converse of this is that the specification of the transition rules themselves are more complicated than those of a reduction system (where some — in particular input and output — rules are combined, and structural congruence abstracts away many of the issues concerning context).

The basic informal concept of the rules is that a process evolves into another process by performing some (*usually* observable) action; for example $\mathrm{P} \xrightarrow{\mu} \mathrm{P}'$ indicates that process P reduces to process P′ by performing action μ as it does so.

Three forms of action are identified (the symbol μ ranges over all three and denotes the anonymous action, for when the precise form is irrelevant):

- τ, the silent (unobservable) action. This indicates that the reduction takes place internally; that is *without interaction with the environment.*
- $x(\overrightarrow{y})$, an input action. The vector $\overrightarrow{y}$ is the sequence of names *received* (not instantiated).
- $(\nu\,\overrightarrow{z})\overline{x}[\overrightarrow{y}]$, an output action. The vector $\overrightarrow{y}$ is the sequence of names emitted during the action; the names $\overrightarrow{z}$ are the subset of $\{x, \overrightarrow{y}\}$ that are *bound*, to account for scope extrusion.

In the two observable actions (input and output) above, x is referred to as the *subject* of the action and $\overrightarrow{y}$ as the *object*. The notation $\|\mu\|$ returns the *subject* of the action μ; that is, $\|(\nu\,\overrightarrow{z})\overline{x}[\overrightarrow{y}]\| = x$ and $\|x(\overrightarrow{y})\| = x$ (and $\|\tau\|$ is undefined).

Definition 2.2.8 *Let the free names of an action in the labelled transition semantics, $FN(\mu)$, be defined as follows:*

- $FN(x(\overrightarrow{y})) = \{x, \overrightarrow{y}\}$
- $FN((\nu\,\overrightarrow{z})\overline{x}[\overrightarrow{y}]) = \{x, \overrightarrow{y}\} - \overrightarrow{z}$
- $FN(\tau) = \emptyset$

The bound names $BN(\mu)$ are defined similarly.

Definition 2.2.9 (Traces) *The multi-step closure of a labelled reduction, ranged over by $\overrightarrow{\mu}$, is referred to as a trace. A multi-step reduction is represented as:*

$$\mathrm{P} \xrightarrow{\overrightarrow{\mu}}\!\!\!\!\rightarrow \mathrm{P}'$$

For a given process P*, the set of all possible traces is given by $Tr(\mathrm{P})$.*

Traces are defined inductively from the single-step relation of Figure 2.1 in Figure 2.2.

$$\frac{\mathrm{P} \xrightarrow{\mu} \mathrm{Q} \quad \mathrm{P} \equiv_\alpha \mathrm{P}'}{\mathrm{P}' \xrightarrow{\mu} \mathrm{Q}} \ (alp.)$$

$$\frac{}{\overline{x}[\overrightarrow{y}].\mathrm{P} \xrightarrow{\overline{x}[\overrightarrow{y}]} \mathrm{P}} \ (out.) \qquad \frac{}{x(\overrightarrow{y}).\mathrm{P} \xrightarrow{x(\overrightarrow{z})} \mathrm{P}\{\overrightarrow{z}/\overrightarrow{y}\}} \ (inp.)$$

$$\frac{\mathrm{P} \xrightarrow{\mu} \mathrm{P}'}{\mathrm{P} + \mathrm{Q} \xrightarrow{\mu} \mathrm{P}'} \ (sum.) \qquad \frac{\mathrm{P} \xrightarrow{\mu} \mathrm{P}' \quad BN(\mu) \cap FN(\mathrm{Q}) = \emptyset}{\mathrm{P}|\mathrm{Q} \xrightarrow{\mu} \mathrm{P}'|\mathrm{Q}} \ (comp.)$$

$$\frac{\mathrm{P} \xrightarrow{\mu} \mathrm{P}' \quad x \notin FN(\mu) \cup BN(\mu)}{(\nu x)\mathrm{P} \xrightarrow{\mu} (\nu x)\mathrm{P}'} \ (res.)$$

$$\frac{\mathrm{P} \xrightarrow{(\nu \overrightarrow{z})\overline{x}[\overrightarrow{y}]} \mathrm{P}' \quad w \neq x, w \in \overrightarrow{y} - \overrightarrow{z}}{(\nu w)\mathrm{P} \xrightarrow{(\nu w, \overrightarrow{z})\overline{x}[\overrightarrow{y}]} \mathrm{P}'} \ (open)$$

$$\frac{\mathrm{P} \xrightarrow{(\nu \overrightarrow{z})\overline{x}[\overrightarrow{y}]} \mathrm{P}' \quad \mathrm{Q} \xrightarrow{x(\overrightarrow{y})} \mathrm{Q}' \quad \overrightarrow{z} \notin FN(\mathrm{Q})}{\mathrm{P}|\mathrm{Q} \xrightarrow{\tau} (\nu \overrightarrow{z})(\mathrm{P}'|\mathrm{Q}')} \ (comm.)$$

Figure 2.1: First-Order Labelled Transition Semantics

$$\frac{}{\mathrm{P} \overset{\tau}{\twoheadrightarrow} \mathrm{P}} \ (reflex) \qquad \frac{\mathrm{P} \equiv_\alpha \mathrm{Q}}{\mathrm{P} \overset{\tau}{\twoheadrightarrow} \mathrm{Q}} \ (alpha.)$$

$$\frac{\mathrm{P} \xrightarrow{\mu} \mathrm{P}' \quad \mathrm{P}' \overset{\overrightarrow{\mu}}{\twoheadrightarrow} \mathrm{P}''}{\mathrm{P} \overset{\mu\overrightarrow{\mu}}{\longrightarrow\!\!\!\!\!\twoheadrightarrow} \mathrm{P}''} \ (trans.)$$

Figure 2.2: First-Order Traces

The rules are presented in Figure 2.1; for space and clarity reasons, the symmetric cases of the rules for summation, composition, and communication are omitted. An additional complication occurs when considering inadvertent capture of names, which was not an issue in the rewrite rules due to the assumption of Convention 2.2.4. In the labelled transition semantics, however, by definition the context is unknown and new names may be imported. This requires the addition of a number of side-conditions to prevent any name capture occurring as the result of a reduction:

- In the (*comp.*) rule the side-condition is necessary to ensure that

  $$x(z).\mathrm{P}|\mathrm{Q} \xrightarrow{x(y)} \mathrm{P}\{y/z\}|\mathrm{Q}$$

  and not

  $$x(z).\mathrm{P}|\mathrm{Q} \xrightarrow{x(y)} (\mathrm{P}|\mathrm{Q})\{y/z\}$$

  where y is also free in Q and is inadvertently captured. See also Convention 2.2.4.

- In the rule (*res.*), the side-condition prevents a name being captured in an input action, for example

  $$(\nu z)x(y).\mathrm{P} \xrightarrow{x(z)} (\nu z)\mathrm{P}\{z/y\}$$

  in which z has become bound (note that this can be avoided by renaming the restricted variable z before the reduction; see also Convention 2.2.4).

- In the rule (*comm.*) the side-condition prevents a name being captured in a communication such as

  $$(\nu z)\overline{x}[y].\mathrm{P} \mid (x(w).\mathrm{Q}|\overline{z}.\mathbf{0}) \xrightarrow{\tau} (\nu z)(\mathrm{P} \mid \mathrm{Q}\{y/w\} \mid \overline{z}.\mathbf{0})$$

  in which the z in $\overline{z}.\mathbf{0}$ has become bound after the reduction.

- The side-condition in the rule (*open.*), together with the conditions on the rules (*comm.*) and (*comp..*), provides the implementation of scope extrusion in the labelled transition semantics by moving a restriction on the term to a restriction in the label.

From now on, these side-conditions (and binders) will be assumed to hold, and will be omitted.

As with the reduction semantics, the Kleene-closure of $\xrightarrow{\mu}$ is written as $\xrightarrow{\overrightarrow{\mu}}\!\!\!\!\!\rightarrow$ (where $\overrightarrow{\mu}$ is a possibly zero-length sequence of labels), and following the convention regarding the single-step reduction, if $\overrightarrow{\mu} = \overrightarrow{\tau}$ (that is, a possibly zero-length sequence of internal reductions) then $\twoheadrightarrow$ will be used in place of $\xrightarrow{\overrightarrow{\tau}}\!\!\!\!\!\rightarrow$.

Equivalence of Reduction and Labelled Transition Semantics

Perhaps not surprisingly (since they have a common goal) it has been demonstrated that there is a strong equivalence between the two forms of operational semantics (Milner 1992). Informally, and modulo structural congruence, the silent labelled transition $\xrightarrow{\tau}$ corresponds exactly to the unlabeled arrow $\rightarrow$ of the reduction semantics. It can also be shown what forms of process in the reduction semantics correspond to the various actions in the labelled semantics.

Consequently, where convenient and unambiguous $\xrightarrow{\tau}$ will sometimes be written just as $\rightarrow$.

2.2.2 Higher-order π-calculus

The growth in distributed computation has seen a natural progression into mobile code: that is, instead of merely propagating data across the network, entire processes or code fragments can be moved to execute in different locations. The advantages of this are manifold; to take a simple example, a web application must validate all data before processing it: this is usually done by the server, and if any errors are found they are reported back to the user for correction. This obviously takes several server transactions to complete, and relies on the server doing all the work: doing at least some of the validation client-side both lessens the load on the server (enabling it to be more efficiently utilised) and speeds the whole transaction for the user, especially if there is significant lag in the network or load on the server.

The concept can be further elaborated: tasks requiring intensive calculation may gain a performance increase by distributing code to execute subtasks across multiple machines that would otherwise remain idle. Examples of this approach already in operation at the time of publication include the `SETI@Home` project (*SETI@home: Search for Extraterrestrial Intelligence at home* 2003) (for analysing radio telescope data) and `distributed.net` (*Distributed.net* 2003) (for number factoring and encryption-cracking); both of these are highly-popular programs that users around the world can run on their desktops (typically as a screensaver, executed when there is idle CPU) to participate in scientific projects requiring massive computational resources.

This is undoubtedly a seductive vision; but also one fraught with danger. Most people, before allowing code from a remote source to execute on their personal machines, would like some assurances that the code in question will do no harm. This combination of both great benefit and risk has naturally led to a lot of research and implementations seeking to tackle the issue.

It is in this setting that the higher-order π-calculus is introduced. The

higher-order π-calculus is a natural extension of the first-order calculus that allows processes to be transmitted as well as channels (that is, processes are first-class data objects). It is thus a natural choice for modelling the issues described above, and as will become apparent later, it reveals some fascinating insights and issues.

Syntax

The complete syntax is shown in Definition 2.2.10:

Definition 2.2.10 *The higher-order syntax is (naturally) very similar to the first-order case (Definition 2.2.2); however some new conventions are first necessitated: The set $\mathcal{A}$ contains all agents* A*; then let K, J range over the set $\mathcal{K} = \mathcal{N} \cup \mathcal{A}$ (that is,* either *an agent or a name); X, Y range over the set of process variables $\mathcal{X}$ (that is, variables that may be instantiated to processes); and V, W range over the set $\mathcal{V} = \mathcal{N} \cup \mathcal{X}$ of names and process variables.*

$$
\begin{array}{lcl}
\mathrm{P} & ::= & \mathbf{0} \mid \sum_{i \in I} \pi_i.\mathrm{P}_i \mid !\mathrm{P} \mid \mathrm{P}|\mathrm{P} \mid (\nu x)\mathrm{P} \mid X \mid X\langle \overrightarrow{K} \rangle \mid \mathrm{F}\langle \overrightarrow{K} \rangle \\
\mathrm{F} & ::= & (\overrightarrow{V})\mathrm{P} \\
\mathrm{A} & ::= & \mathrm{P} \mid \mathrm{F} \\
\pi & ::= & x(\overrightarrow{V}) \mid \overline{x}[\overrightarrow{V}]
\end{array}
$$

Most of this is familiar from Section 2.2.1 (such as null, composition and summation, and replication), however the new constructs deserve some informal coverage. First, note that a process may now also be a *variable* (X): this is a natural consequence of the fact that processes may now be transmitted, so there must be a way to bind them at the receiving end. Following this observation, note that the output case now specifies $\overrightarrow{K}$ as arguments; in other words, either names or processes may be output (as required). Similarly, the bound parameters in an input construct ($\overrightarrow{V}$) may be either a name or a variable, in symmetry with the output construct.

What is quite different is the introduction of an *abstraction* construct (ranged over by F), and the corresponding *application* forms ($\mathrm{F}\langle \mathrm{K} \rangle$ and $X\langle \overrightarrow{K} \rangle$). The introduction of these complementary forms arises from the need — in a true higher-order context — to be able to *parameterise* processes received with respect to a given name or process.

Agents A, B are either processes or abstractions.

Definition 2.2.11 (Higher-Order Substitutions) *Higher order term substitution is defined similarly as for the first-order calculus (see Definition 2.2.6). For a given term* B*, write* $\mathrm{B}\{A/X\}$ *to denote the same term with all free occurrences of* X *replaced by* A*. The definition is extended in the usual way to cover sequences, for example* $\mathrm{B}\{\overrightarrow{\mathrm{A}}/\overrightarrow{X}\}$*.*

More formally, substitutions may be defined inductively:

$$\begin{array}{rcl}
\mathbf{0}\{\mathrm{A}/X\} & \triangleq & \mathbf{0} \\
X\{\mathrm{A}/X\} & \triangleq & \mathrm{A} \\
Y\{\mathrm{A}/X\} & \triangleq & Y \quad (Y \neq X) \\
!\mathrm{P}\{\mathrm{A}/X\} & \triangleq & !(\mathrm{P}\{\mathrm{A}/X\}) \\
(\mathrm{P}|\mathrm{Q})\{\mathrm{A}/X\} & \triangleq & (\mathrm{P}\{\mathrm{A}/X\}) \mid (\mathrm{Q}\{\mathrm{A}/X\}) \\
(\Sigma_i \mathrm{P}_i)\{\mathrm{A}/X\} & \triangleq & \Sigma_i(\mathrm{P}_i\{\mathrm{A}/X\}) \\
(\nu x)\mathrm{P}\{\mathrm{A}/X\} & \triangleq & (\nu x)(\mathrm{P}\{\mathrm{A}/X\}) \\
(x(\overrightarrow{V}).\mathrm{P})\{\mathrm{A}/X\} & \triangleq & x(\overrightarrow{V}).(\mathrm{P}\{\mathrm{A}/X\}) \\
(\overline{x}[\overrightarrow{K}].\mathrm{P})\{\mathrm{A}/X\} & \triangleq & \overline{x}[\overrightarrow{K}\{\mathrm{A}/X\}].(\mathrm{P}\{\mathrm{A}/X\}) \\
X\langle K\rangle\{\mathrm{A}/X\} & \triangleq & \mathrm{P}\{(\overrightarrow{K}\{\mathrm{A}/X\})/\overrightarrow{V}\} \quad (\mathrm{A} \equiv (\overrightarrow{V})\mathrm{P}) \\
(Y\langle\overrightarrow{K}\rangle)\{\mathrm{A}/X\} & \triangleq & Y\langle(\overrightarrow{K}\{\mathrm{A}/X\})\rangle \quad (Y \neq X) \\
(\mathrm{F}\langle\overrightarrow{K}\rangle)\{\mathrm{A}/X\} & \triangleq & (\mathrm{F}\{\mathrm{A}/X\})\langle\overrightarrow{K}\{\mathrm{A}/X\}\rangle
\end{array}$$

As before, if $\overrightarrow{K}$ *represents the sequence* $K_1 \ldots K_n$ *then* $\overrightarrow{K}\{\mathrm{A}/X\}$ *represents* $K_1\{\mathrm{A}/X\} \ldots K_n\{\mathrm{A}/X\}$*. Convention 2.2.4 applies in the higher-order calculus as well. Name substitution is handled identically to the first-order case in Definition 2.2.6 (page 21).*

Reduction Semantics

The reduction semantics of the higher-order calculus are in essence no different from the first-order case; the communication case is merely changed to specify process as well as data transmission:

$$\frac{}{(\ldots + \overline{x}[\overrightarrow{K}].\mathrm{P}|x(\overrightarrow{V}).\mathrm{Q} + \ldots) \rightarrow \mathrm{P}|\mathrm{Q}\{\overrightarrow{K}/\overrightarrow{V}\}}$$

Application of abstractions is handled with addition to the definition of structural congruence (Definition 2.2.5):

$$(\overrightarrow{V})\mathrm{P}\langle\overrightarrow{K}\rangle \equiv \mathrm{P}\{\overrightarrow{K}/\overrightarrow{V}\}$$

The induction rules are unchanged.

Labelled Transition Semantics

The labelled transition semantics are also largely unchanged; the two observable actions need to be altered to accommodate the possibility of communication of processes (and instantiation of variables to processes), and the restriction rule specifies that both a name and a variable may be made local. An additional rule is also added to handle reduction of applications. Therefore letting the input action have the form $\xrightarrow{x(\overrightarrow{K})}$ and the output action $\xrightarrow{(\nu\overrightarrow{V})\overline{x}[\overrightarrow{K}]}$ (and μ ranging over the three including the silent action as before), the complete higher-order rules are presented in Figure 2.3. As before, where clear the arrows $\xrightarrow{\tau}$ and $\rightarrow$ (as well as $\xrightarrow{\overrightarrow{\tau}}$ and $\twoheadrightarrow$) will be used interchangeably.

$$\frac{\mathrm{P} \xrightarrow{\mu} \mathrm{Q} \quad \mathrm{P} \equiv_\alpha \mathrm{P}'}{\mathrm{P}' \xrightarrow{\mu} \mathrm{Q}}\ (alp.) \qquad \frac{\mathrm{P}\{\overrightarrow{K}/\overrightarrow{V}\} \xrightarrow{\mu} \mathrm{P}'}{(\overrightarrow{V})\mathrm{P}\langle\overrightarrow{K}\rangle \xrightarrow{\mu} \mathrm{P}'}\ (app.)$$

$$\frac{}{\overline{x}[\overrightarrow{K}].\mathrm{P} \xrightarrow{\overline{x}[\overrightarrow{K}]} \mathrm{P}}\ (out.) \qquad \frac{}{x(\overrightarrow{V}).\mathrm{P} \xrightarrow{x(\overrightarrow{K})} \mathrm{P}\{\overrightarrow{K}/\overrightarrow{V}\}}\ (inp.)$$

$$\frac{\mathrm{P} \xrightarrow{\mu} \mathrm{P}'}{\mathrm{P} + \mathrm{Q} \xrightarrow{\mu} \mathrm{P}'}\ (sum.) \qquad \frac{\mathrm{P} \xrightarrow{\mu} \mathrm{P}'}{\mathrm{P}|\mathrm{Q} \xrightarrow{\mu} \mathrm{P}'|\mathrm{Q}}\ (comp.)$$

$$\frac{\mathrm{P} \xrightarrow{\mu} \mathrm{P}' \quad x \notin FN(\mu) \cup BN(\mu)}{(\nu x)\mathrm{P} \xrightarrow{\mu} (\nu x)\mathrm{P}'}\ (res.)$$

$$\frac{\mathrm{P} \xrightarrow{(\nu\overrightarrow{V})\overline{x}[\overrightarrow{K}]} \mathrm{P}' \quad W \neq x, W \in FN\left(\overrightarrow{K}\right) - \overrightarrow{V}}{(\nu W)\mathrm{P} \xrightarrow{(\nu W\overrightarrow{V})\overline{x}[\overrightarrow{K}]} \mathrm{P}'}\ (open.)$$

$$\frac{\mathrm{P} \xrightarrow{(\nu\overrightarrow{V})\overline{x}[\overrightarrow{K}]} \mathrm{P}' \quad \mathrm{Q} \xrightarrow{x(\overrightarrow{K})} \mathrm{Q}' \quad \overrightarrow{V} \notin FN(\mathrm{Q})}{\mathrm{P}|\mathrm{Q} \xrightarrow{\tau} \mathrm{P}'|\mathrm{Q}'}\ (comm.)$$

Figure 2.3: Higher-Order Labelled Transition Semantics

2.2.3 The need for a Higher-order π-calculus

Davide Sangiorgi (Sangiorgi 1993b) demonstrated a result that is perhaps surprising at first glance, showing that the higher-order π-calculus could be satisfactorily (that is, reduction semantics and congruence relationships are preserved) encoded in the first-order calculus. The key to understanding why this should be the case is to realise that the π-calculus encodes no notion of *physical* locality; that is, processes may be situated on the same computer or on the other side of the world — this detail is abstracted away.[1]

A process is "located" in some sense by its outer-most prefix — if a process in parallel does not possess this name it is effectively blind to the other's existence (and thus, might as well be in a separate location). It is easy to see therefore that a higher-order π-calculus may be simulated by creating a fresh name and locating the process at this name, then sending the name instead of the process — the process receiving this name may then activate the other process at will, now that it is aware of its location (an analogy may be drawn to pointers in programming languages such as C: sending the address rather than the object). This turns out to be precisely what happens. Take for example the case of a process P sending a second process Q, to a third process R which then executes it in parallel:

$$\overline{x}[\mathrm{Q}].\mathrm{P}|x(X).(\mathrm{R}|X) \;\rightarrow\; \mathrm{P}|\mathrm{R}|\mathrm{Q}$$

The same effect (including reduction, albeit in a few more steps) may be achieved as follows:

$$\begin{aligned}(\nu y)(\overline{x}[y].\mathrm{P}|y.\mathrm{Q}) \mid x(z).(\mathrm{R}|\overline{z}.\mathbf{0}) &\rightarrow (\nu y)(\overline{x}[y].\mathrm{P} \mid y.\mathrm{Q} \mid x(z).(\mathrm{R} \mid \overline{z}.\mathbf{0})) \\ &\rightarrow (\nu y)(\mathrm{P} \mid y.\mathrm{Q} \mid \mathrm{R} \mid \overline{y}.\mathbf{0}) \\ &\rightarrow (\nu y)(\mathrm{P} \mid \mathrm{Q} \mid \mathrm{R} \mid \mathbf{0}) \\ &\rightarrow (\nu y)(\mathrm{P} \mid \mathrm{Q} \mid \mathrm{R}) \\ &\rightarrow \mathrm{P} \mid \mathrm{Q} \mid \mathrm{R}\end{aligned}$$

(assuming of course $y \notin FN(\mathrm{P}|\mathrm{Q}|\mathrm{R})$).

The compilation algorithm will not be reproduced here; the interested reader is referred to Sangiorgi (1993b) for the details. Suffice to say that it

[1] This is quite a powerful abstraction because it enables a wide range of networks and programs to be modelled in the same manner; however it is also a "leaky" abstraction in other senses. For instance, there is little or no concept of the time-lag that may exist in wide-area networks, and concepts such as failure and process hiding — for example, behind firewalls — can only be encoded rather unsatisfactorily. For examples of research into calculi designed to overcome these short-comings, see Cardelli and Gordon (1998) and Sewell (1998).

preserves both typing and most useful notions of congruence and bisimulation.

The question may well be asked then: is there any point in examining a separate higher-order calculus if the same results may be obtained in the first-order calculus? The author feels that the main reason it is still worthy of analysis is clarity, after all, it is certainly *possible* to encode numerals in the λ-calculus, but very few people program in this manner! None the less, it is not just a matter of syntactic sugar; it would be reasonable to expect that many analyses and subtle points may present themselves examining a higher-order calculus "natively" that would not be readily apparent working with a first-order encoding. It is even possible that some more sophisticated type systems or analyses not convertible to a first-order equivalent may present themselves. In fact, Sangiorgi (1993b) notes several examples that gain considerable elegance (or would possibly have been overlooked had a first-order calculus been used) due to the use of a higher-order calculus. Vivas and Yoshida (2002) also notes that in a widely-distributed setting (not just a local-area network, for example) there are considerable differences between migrating and merely activating code that may not be satisfactorily represented with the first-order encoding outlined above.

2.3 Type Systems for the π-calculus

The focus now shifts on to what will form the core emphasis of this thesis: type systems for the π-calculus. Type systems traditionally seek to establish some kind of "well-formedness" properties, and to guarantee that any program that is well-formed with respect to that particular system will never encounter certain types of run-time errors.

The most immediate (or probable) cause of run-time error in even the simplest form of the polyadic π-calculus is arity mismatch, whereby a process transmits a certain number of names to a process expecting to receive a different number of names. In an example such as

$$\overline{x}[wy].\mathrm{P} \mid x(z).\mathrm{Q}$$

it is easy to detect; however the situation can rapidly become complicated. Take for example:

$$y(z).\Big(\overline{z}[wy].\mathbf{0} \mid x(y').\mathrm{Q}\Big) \mid \overline{y}[x].\mathrm{P}$$

This appears innocuous at first glance, but reduces (assuming z is not free in Q) to $\overline{x}[wy].\mathbf{0} \mid x(y').\mathrm{Q}$ which again has a clear arity mismatch. Any

type scheme must therefore prevent such errors from occurring. If primitive data-types are admitted to the calculus it is also reasonable to expect that constraints on their usage would be enforced; for example channel x may only be used to carry $\langle int, bool \rangle$ pairs; however this angle will not be pursued any further (it is trivial to add, but the focus here is restricted to the core calculus).

2.3.1 Types and Judgements

As a notational shorthand, $\overrightarrow{x} : \overrightarrow{\sigma}$ will often be written to mean $x_1 : \sigma_1, \ldots, x_n : \sigma_n$; assuming that $\overrightarrow{x}$ is $x_1 \ldots x_n$ and $\overrightarrow{\sigma}$ is $\sigma_1 \ldots \sigma_n$ (the sequences are assumed to be of equal length when this shorthand is used).

Definition 2.3.1 (Type Environments)

1. *Type* environments*, ranged over by* Γ *and* Θ*, are partial functions from names to types. As usual,* $\Gamma(x) = \sigma$ *means that* $\langle x, \sigma \rangle \in \Gamma$.

2. *The notation* $\Gamma \asymp \Theta$ *states that for every name* x *in the domain of both* Γ *and* Θ *then* $\Gamma(x) = \Theta(x)$.

3. *Write* Γ, Θ *for the union (*$\Gamma \cup \Theta$*) of the two; note that this is only defined when* $\Gamma \asymp \Theta$*. As further shorthand write* $\Gamma, x : \sigma$ *for the function which is the union of* Γ *and* $\{\langle x, \sigma \rangle\}$ *(the notation* $x : \sigma$ *is used in preference to* $x \mapsto \sigma$ *or* $\langle x, \sigma \rangle$*).*

4. *The function (referred to as an "environment" from now on)* Γ_x *represents* Γ *with* x *removed from its domain.*

Definition 2.3.2 (Judgements) *Several forms of judgement will be used, with other forms necessary for some of the more sophisticated type systems being introduced later, when appropriate.*

A well-formed process judgement is an expression of either of the forms

$$\Gamma \quad \vdash_{\pi} \quad \mathrm{P} : \mathsf{Proc}$$
$$\Gamma \quad \vdash_{(\pi)} \quad \mathrm{P} : \mathsf{Proc}$$

The first states that, for a term P *with free names mapped to types as specified by the environment* Γ*, then* P *is well-formed under the type system in question.*[2] *The second form is identical but only used by null and guarded*

[2]It would probably be clearer in this case to simply write $\Gamma \vdash \mathrm{P}$ and omit the Proc declaration. The judgement form used here is however retained for consistency with the higher-order system, where it is necessary to distinguish between judgements involving well-formed processes and processes abstracted on names or variables, in which case some notion of a process type is required.

processes and summations, in order to enforce the syntactical requirement that only guarded processes may occur in a summation. This is isomorphic to Milner's notion of Normal Processes (Milner 1993). As a further notational shorthand, where convenient write $\overrightarrow{\Gamma} \vdash_\pi \overrightarrow{\mathrm{P}} : \mathsf{Proc}$ *for the series of judgements* $\Gamma_1 \vdash_\pi \mathrm{P}_1 : \mathsf{Proc}, \quad \ldots, \quad \Gamma_n \vdash_\pi \mathrm{P}_n : \mathsf{Proc}$*, where* $\overrightarrow{\Gamma}$ *represents* $\Gamma_1 \ldots \Gamma_n$ *and* $\overrightarrow{\mathrm{P}}$ *represents* $\mathrm{P}_1 \ldots \mathrm{P}_n$ *(also assuming where this shorthand is used that* $\Gamma_1 \asymp \ldots \asymp \Gamma_n$*).*

A type judgement is an expression of the form $\Gamma \vdash_\pi x : \sigma$*, which states that* $\Gamma(x) = \sigma$*. The vector notation may also be used in this case, for example* $\overrightarrow{\Gamma} \vdash_\pi \overrightarrow{x} : \overrightarrow{\sigma}$ *or* $\Gamma \vdash_\pi \overrightarrow{x} : \overrightarrow{\sigma}$*.*

Where applicable, $\bigcup \overrightarrow{\Gamma}$ *represents the union of all elements in the sequence* $\overrightarrow{\Gamma}$*; that is, if* $\overrightarrow{\Gamma}$ *is the sequence* $\Gamma_1 \ldots \Gamma_2$ *then* $\bigcup \overrightarrow{\Gamma} = \Gamma_1 \cup \ldots \cup \Gamma_n$ *(this notation will be used to refer to the union of any sequence, not just environments).*

In all cases when the vector ($\overrightarrow{\cdot}$) notation is used, this may also represent the empty sequence (including a zero-length series of deductions).

2.3.2 First-order Type Systems

So, the type discipline must enforce arity correctness; this perhaps implies that some numerical component will suffice, however it soon becomes apparent that it must also supply information about each *type* carried (a kind of recursive definition). For example, in the process $\overline{x}[yz].\mathrm{P}$, the type of x must not only state that it is used to communicate two names, but also that those names have the types of y and z respectively (and in that order). That is, if y has type σ_y and z type σ_z then a concise way of expressing this information is to write the type of x as $(\sigma_y \sigma_z)$. However, the types of y and z will also have a similar structure; so if it is assumed that y is merely used as a signaling channel and carries no data itself (that is, $\sigma_y = ()$) and that z is used to transmit a single channel, also used just for signaling (so $\sigma_z = (())$), then the complete type of x must be $(()(()))$. (This is not easy to read without some concentration; the complete structure of types will not normally be exposed in this manner).

Definition 2.3.3 (Type Syntax) *Let* σ *range over base types, and where appropriate write examples such as* $(\sigma_1 \ldots \sigma_n)$ *to represent the type of a channel that carries n names, with types* $\sigma_1 \ldots \sigma_n$ *respectively. As mentioned in Section 2.3.1, this may also be written as the more concise* $(\overrightarrow{\sigma})$*. Formally:*

$$\sigma \quad ::= \quad (\overrightarrow{\sigma})$$

To avoid ambiguity during type reconstruction, it is necessary to incorporate information about bound names into the language itself. The typed language syntax is shown in Definition 2.3.4:

Definition 2.3.4 (Typed First-Order Syntax) *The first-order syntax of Definition 2.2.2 is extended with type information for bound names as shown:*

$$\begin{aligned} \mathrm{P} &::= \mathbf{0} \mid \sum_{i \in I} \pi_i.\mathrm{P}_i \mid \mathrm{P}|\mathrm{P} \mid !\mathrm{P} \mid (\nu x : \sigma)\mathrm{P} \\ \pi &::= \overline{x}[\overrightarrow{y}] \mid x(\overrightarrow{y} : \overrightarrow{\sigma}) \end{aligned}$$

All other details concerning meta-variables and so on remain the same, for example processes are still ranged over by P*,* Q*, and* R*.*

The complete rules for well-formed first-order processes are shown in Figure 2.4. They are mostly straight-forward, ensuring that only well-formed processes may be combined (the summation and composition rules), and that communication constructs (input and output) are only formed with suitably typed channels. The different judgement forms used (see Definition 2.3.2) guarantee that only guarded and null processes may be used in a summation, while the rule *conv.* ensures that any process may be used in a composition or as a base for input, output, restriction, or replication. The restriction rule removes the restricted name from the environment, as it is a binding rule. The most basic well-formed process is the null, or inactive, process. This is well-formed under any environment, thus incorporating weakening into the type system.

Example 2.3.5 *The following judgement is derivable:*

$$x : (()(())), y : (), z : (()) \vdash_\pi \overline{x}[yz].\mathbf{0} : \mathsf{Proc}$$

First-Order Typed Labelled Transition Semantics

A variant of the labelled transition semantics for the first-order calculus may be given that incorporates the type system. This offers little additional insight for the plain calculus, but will prove useful in understanding the extensions presented later so the basic version is described here.

Definition 2.3.6 *A reduction in the typed labelled transition semantics has the following format:*

$$\langle \Theta, \Gamma, \mathrm{P} \rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}' \rangle$$

This states that for the judgement $\Gamma \vdash_\pi \mathrm{P} : \mathsf{Proc}$ *the process* P *may communicate with a second anonymous process which is well-typed under environment*

$$\frac{}{\Gamma \vdash_{(\pi)} \mathbf{0} : \mathsf{Proc}}\ (null.) \qquad \frac{\Gamma \vdash_{(\pi)} \mathrm{P} : \mathsf{Proc}}{\Gamma \vdash_{\pi} \mathrm{P} : \mathsf{Proc}}\ (conv.)$$

$$\frac{\Gamma_x, x : \sigma \vdash_{\pi} \mathrm{P} : \mathsf{Proc}}{\Gamma_x \vdash_{\pi} (\nu x : \sigma)\mathrm{P} : \mathsf{Proc}}\ (res.) \qquad \frac{\Gamma \vdash_{\pi} \mathrm{P} : \mathsf{Proc} \quad \Theta \vdash_{\pi} \mathrm{Q} : \mathsf{Proc}}{\Gamma, \Theta \vdash_{\pi} \mathrm{P}|\mathrm{Q} : \mathsf{Proc}}\ (comp.)$$

$$\frac{\Gamma \vdash_{\pi} \mathrm{P} : \mathsf{Proc}}{\Gamma \vdash_{\pi} !\mathrm{P} : \mathsf{Proc}}\ (rep.) \qquad \frac{\Gamma \vdash_{(\pi)} \mathrm{P} : \mathsf{Proc} \quad \Theta \vdash_{(\pi)} \mathrm{Q} : \mathsf{Proc}}{\Gamma, \Theta \vdash_{(\pi)} \mathrm{P} + \mathrm{Q} : \mathsf{Proc}}\ (sum.)$$

$$\frac{\Gamma_{\overrightarrow{y}}, x : (\overrightarrow{\sigma}), \overrightarrow{y} : \overrightarrow{\sigma} \vdash_{\pi} \mathrm{P} : \mathsf{Proc}}{\Gamma_{\overrightarrow{y}}, x : (\overrightarrow{\sigma}) \vdash_{(\pi)} x(\overrightarrow{y} : \overrightarrow{\sigma}).\mathrm{P} : \mathsf{Proc}}\ (inp.)$$

$$\frac{\Gamma, x : (\overrightarrow{\sigma}), \overrightarrow{y} : \overrightarrow{\sigma} \vdash_{\pi} \mathrm{P} : \mathsf{Proc}}{\Gamma, x : (\overrightarrow{\sigma}), \overrightarrow{y} : \overrightarrow{\sigma} \vdash_{(\pi)} \overline{x}[\overrightarrow{y}].\mathrm{P} : \mathsf{Proc}}\ (out.)$$

Figure 2.4: First-Order π-calculus Type Rules

Θ*, performing the action* μ *and transforming itself to* P'*, its environment to* Γ'*, and the environment of the second process to* Θ'*. It is assumed that the two environments are compatible, that is* $\Gamma \asymp \Theta$*.*

The complete semantics is presented in Figure 2.5. They are mostly unchanged from their plain version, but include additional information describing the names that may be acquired by an environment during an input. The case for communication is notable as it is the only one that constructs an internal action. For this reason the environments in the two reductions in the antecedents are the dual of each other, as the external environment that one process interacts with is the environment of the other, and in the consequent the two are merged and an arbitrary external environment Δ is used which remains unchanged by the internal action.

2.3.3 Higher-order Type Systems

As may be expected, the type rules for the higher-order calculus differ very little from the first-order rules, however there is some subtlety that requires examination. The complicating factor is the need to distinguish between binding to a name and binding to a variable; a name may only be bound to another name, and likewise only a process may be bound to a variable.

$$\frac{\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle \quad \mathrm{P} \equiv_\alpha \mathrm{Q}}{\langle \Theta, \Gamma, \mathrm{Q}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle}\ (alp.)$$

$$\frac{}{\langle \Theta, \Gamma \cup \{\overrightarrow{y} : \overrightarrow{\sigma}\}, \overline{x}[\overrightarrow{y}].\mathrm{P}\rangle \xrightarrow{\overline{x}[\overrightarrow{y}]} \langle \Theta \cup \{\overrightarrow{y} : \overrightarrow{\sigma}\}, \Gamma \cup \{\overrightarrow{y} : \overrightarrow{\sigma}\}, \mathrm{P}\rangle}\ (out.)$$

$$\frac{}{\langle \Theta \cup \{\overrightarrow{y} : \overrightarrow{\sigma}\}, \Gamma, x(\overrightarrow{z} : \overrightarrow{\sigma}).\mathrm{P}\rangle \xrightarrow{x(\overrightarrow{y})} \langle \Theta \cup \{\overrightarrow{y} : \overrightarrow{\sigma}\}, \Gamma \cup \{\overrightarrow{y} : \overrightarrow{\sigma}\}, \mathrm{P}\{\overrightarrow{y}/\overrightarrow{z}\}\rangle}\ (inp.)$$

$$\frac{\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle}{\langle \Theta, \Gamma, \mathrm{P} + \mathrm{Q}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle}\ (sum.)$$

$$\frac{\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle \quad BN(\mu) \cap FN(\mathrm{Q}) = \emptyset}{\langle \Theta, \Gamma, \mathrm{P}|\mathrm{Q}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'|\mathrm{Q}\rangle}\ (comp.)$$

$$\frac{\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle}{\langle \Theta, \Gamma, !\mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'|!\mathrm{P}\rangle}\ (rep.)$$

$$\frac{\langle \Theta, \Gamma_x \cup \{x : \sigma\}, \mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma'_x \cup \{x : \sigma\}, \mathrm{P}'\rangle \quad x \notin FN(\mu) \cup BN(\mu)}{\langle \Theta_x, \Gamma_x, (\nu x : \sigma)\mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta'_x, \Gamma'_x, (\nu x : \sigma)\mathrm{P}'\rangle}\ (res.)$$

$$\frac{\langle \Theta, \Gamma_w \cup \{w : \sigma\}, \mathrm{P}\rangle \xrightarrow{(\nu\overrightarrow{z})\overline{x}[\overrightarrow{y}]} \langle \Theta', \Gamma'_w \cup \{w : \sigma\}, \mathrm{P}'\rangle \quad w \neq x,\ w \in \overrightarrow{y} - \overrightarrow{z}}{\langle \Theta_w, \Gamma_w, (\nu w : \sigma)\mathrm{P}\rangle \xrightarrow{(\nu w\overrightarrow{z})\overline{x}[\overrightarrow{y}]} \langle \Theta'_w, \Gamma'_w, (\nu w : \sigma)\mathrm{P}'\rangle}\ (open)$$

$$\frac{\begin{array}{l}\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{(\nu\overrightarrow{z})\overline{x}[\overrightarrow{y}]} \langle \Theta', \Gamma', \mathrm{P}'\rangle \\ \langle \Gamma, \Theta, \mathrm{Q}\rangle \xrightarrow{x(\overrightarrow{y})} \langle \Gamma', \Theta', \mathrm{Q}\rangle \qquad \overrightarrow{z} \notin FN(\mathrm{Q})\end{array}}{\langle \Delta, \Gamma \cup \Theta, \mathrm{P}|\mathrm{Q}\rangle \xrightarrow{\tau} \langle \Delta, \Gamma' \cup \Theta', \mathrm{P}'|\mathrm{Q}'\rangle}\ (comm.)$$

Figure 2.5: First-Order Typed Labelled Transition Semantics

For this reason, the type syntax is expanded slightly in order to be able to distinguish between a process and a name (Definition 2.3.7):

Definition 2.3.7 (Higher-Order Type Syntax)

$$\sigma \quad ::= \quad (\overrightarrow{\sigma}) \mid \mathsf{Proc} \mid (\overrightarrow{\sigma}) \rightarrow \mathsf{Proc}$$

An ordinary channel type is the same as before. A process type is introduced to distinguish the type of a variable from the type of a name. As a process may be paramaterised (abstracted), it is also necessary to specify the types of names and processes upon which it is parameterised. As before a vector can include the empty vector (that is, a channel not used for data transmission, or a thunked process $() \rightarrow \mathsf{Proc}$*).*

Definition 2.3.8 *The higher-order language is also extended in a similar manner to Definition 2.3.4 to include types on bound names and variables:*

$$\begin{array}{lcl} \mathrm{P} & ::= & \mathbf{0} \mid \sum_{i \in I} \pi_i.\mathrm{P}_i \mid !\mathrm{P} \mid \mathrm{P}|\mathrm{P} \mid (\nu x : \sigma)\mathrm{P} \mid X \mid X\langle \overrightarrow{K} \rangle \mid \mathrm{F}\langle \overrightarrow{K} \rangle \\ \mathrm{F} & ::= & (\overrightarrow{V} : \overrightarrow{\sigma})\mathrm{P} \\ \mathrm{A} & ::= & \mathrm{P} \mid \mathrm{F} \\ \pi & ::= & x(\overrightarrow{V} : \overrightarrow{\sigma}) \mid \overline{x}[\overrightarrow{V}] \end{array}$$

The higher-order type rules themselves are equally predictable, and are presented in Figure 2.6. The null, replication, restriction, composition, and summation rules are exactly as they appeared before in Figure 2.4. The input and output rules are also similar; the only difference is they are now more general and refer to either variables *or* names as appropriate. There is a new variable introduction axiom that is largely motivated by the new form of the output rule: since there is now the ability to transmit *processes* as well as names, the operands must be declared on the *right-hand side* of the turn-style (since processes cannot be admitted on the left-hand side). It is thus much more convenient to be able to introduce names as well as variables and processes on the right-hand side, and exploit the syntactic shorthand form $\overrightarrow{\Gamma} \vdash_\pi \overrightarrow{K} : \mathsf{Proc}\overrightarrow{\sigma}$.

Higher-Order Typed Labelled-Transition-Semantics

The typed labelled-transition-semantics for the higher-order calculus can be described in a similar manner to those for the first-order; they are presented in Figure 2.7. The differences from the first-order labelled-transition-semantics

$$\frac{}{\Gamma \vdash_{(\pi)} \mathbf{0} : \mathsf{Proc}}\ (null) \qquad \frac{}{\Gamma, V : \sigma \vdash_{\pi} V : \sigma}\ (var.)$$

$$\frac{\Gamma \vdash_{(\pi)} \mathrm{P} : \mathsf{Proc}}{\Gamma \vdash_{\pi} \mathrm{P} : \mathsf{Proc}}\ (conv.) \qquad \frac{\Gamma \vdash_{\pi} \mathrm{P} : \mathsf{Proc} \quad \Theta \vdash_{\pi} \mathrm{Q} : \mathsf{Proc}}{\Gamma, \Theta \vdash_{\pi} \mathrm{P}|\mathrm{Q} : \mathsf{Proc}}\ (comp.)$$

$$\frac{\Gamma \vdash_{\pi} \mathrm{P} : \mathsf{Proc}}{\Gamma \vdash_{\pi} !\mathrm{P} : \mathsf{Proc}}\ (repl.) \qquad \frac{\Gamma \vdash_{(\pi)} \mathrm{P} : \mathsf{Proc} \quad \Theta \vdash_{(\pi)} \mathrm{Q} : \mathsf{Proc}}{\Gamma, \Theta \vdash_{(\pi)} \mathrm{P} + \mathrm{Q} : \mathsf{Proc}}\ (sum.)$$

$$\frac{\Gamma_x, x : \sigma \vdash_{\pi} \mathrm{P} : \mathsf{Proc}}{\Gamma_x \vdash_{\pi} (\nu x : \sigma)\mathrm{P} : \mathsf{Proc}}\ (res.) \qquad \frac{\Gamma_{\overrightarrow{V}}, \overrightarrow{V} : \overrightarrow{\sigma} \vdash_{\pi} \mathrm{P} : \mathsf{Proc}}{\Gamma_{\overrightarrow{V}} \vdash_{\pi} (\overrightarrow{V} : \overrightarrow{\sigma})\mathrm{P} : (\overrightarrow{\sigma}) \to \mathsf{Proc}}\ (abs.)$$

$$\frac{\Gamma \vdash_{\pi} \mathrm{A} : (\overrightarrow{\sigma}) \to \mathsf{Proc} \quad \overrightarrow{\Theta} \vdash_{\pi} \overrightarrow{K} : \overrightarrow{\sigma}}{\Gamma, \overrightarrow{\Theta} \vdash_{\pi} \mathrm{A}\langle \overrightarrow{K} \rangle : \mathsf{Proc}}\ (app.)$$

$$\frac{\Gamma_{\overrightarrow{V}}, x : (\overrightarrow{\sigma}), \overrightarrow{V} : \overrightarrow{\sigma} \vdash_{\pi} \mathrm{P} : \mathsf{Proc}}{\Gamma_{\overrightarrow{V}}, x : (\overrightarrow{\sigma}) \vdash_{(\pi)} x(\overrightarrow{V} : \overrightarrow{\sigma}).\mathrm{P} : \mathsf{Proc}}\ (inp.)$$

$$\frac{\Gamma, x : (\overrightarrow{\sigma}) \vdash_{\pi} \mathrm{P} : \mathsf{Proc} \quad \overrightarrow{\Theta} \vdash_{\pi} \overrightarrow{K} : \mathsf{Proc}\,\overrightarrow{\sigma}}{\Gamma, x : (\overrightarrow{\sigma}), \overrightarrow{\Theta} \vdash_{(\pi)} \overline{x}[\overrightarrow{K}].\mathrm{P} : \mathsf{Proc}}\ (out.)$$

Figure 2.6: Higher-Order π-calculus Type Rules

in Figure 2.5 are slight: there is a case for application (rule *app.*), and communications can contain agents as well as names. This last difference necessitates a small change in presentation of rules such as those for input and output as agents cannot be declared on the left side of the turn-style, but the intentions are the same. Note the strict requirements to ensure that the environments used in typing the objects of the communication are a subset of the main environment, and contain exactly the free names of the names and agents communicated. This ensures that no extraneous names are passed to the other process environment.

2.3.4 Sortings

It is worth mentioning that the type systems presented here (which form the basis for the rest of this thesis) are not the only approach to ensuring run-time safety in the π-calculus (first- or higher-order); the original presentations (see Milner et al. 1992a,b; Milner 1993) in fact used a discipline of *sortings*. The end result is similar, but the methodology is slightly different: each name in the calculus is associated (via an environment) with a *sort* (an *atomic* variable), then a separate environment associates each sort with a list of other sorts (possibly including itself) representing the arity and range of names that may be transmitted by all names having that sort. Note that this has two benefits: firstly, it greatly simplifies the treatment of recursive structures since recursion is inherit in the sorting (rather than the typing structure), and secondly, it enables a finer-grained distinction between sorts than allowed by types based on the ability to differentiate based on *name* as well as structure.

An example should be sufficient to illustrate these properties (Example 2.3.9):

Example 2.3.9 *Let sorts be ranged over by ρ; as before Γ is an environment (function) mapping free names to sorts. An additional environment called* Ob *maps sorts to their objects (that is, sorts carried by that sort).*

For example, if $\Gamma = \{x : \rho_1, y : \rho_2, z : \rho_3\}$ and the program $\overline{x}[yz].\mathbf{0}$ is to be well-formed under $\Gamma;\mathrm{Ob}$, then Ob *must contain $\rho_1 \mapsto (\rho_2\rho_3)$. Note that as sorts are atomic, with their behaviour/structure determined by* Ob*, then it is possible to distinguish ρ_1 from ρ_4 where $\rho_4 \mapsto (\rho_2\rho_3) \in \mathrm{Ob}$, even though their structure is identical.*

An intuitive treatment of recursion is also possible using sorts, for example if $\rho_1 \mapsto (\rho_1\rho_2) \in \mathrm{Ob}$ then $\overline{x}[xy]$ is well formed.

No further mention of sortings (or recursion) is made here; the interested reader is referred to an earlier version of this work that utilised a sorting

$$\frac{\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle \quad \mathrm{P} \equiv_\alpha \mathrm{Q}}{\langle \Theta, \Gamma, \mathrm{Q}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle}\ (alp.)$$

$$\frac{\langle \Theta, \Gamma, \mathrm{P}\{\overrightarrow{K}/\overrightarrow{V}\}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle}{\langle \Theta, \Gamma, (\overrightarrow{V})\mathrm{P}\langle\overrightarrow{K}\rangle\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle}\ (app.)$$

$$\frac{\overrightarrow{\Gamma'} \vdash_\pi \overrightarrow{K} : \overrightarrow{\sigma} \quad \bigcup \overrightarrow{\Gamma'} \subseteq \Gamma \quad \mathrm{dom} \bigcup \overrightarrow{\Gamma'} = FN\left(\overrightarrow{K}\right)}{\langle \Theta, \Gamma, \overline{x}[\overrightarrow{K}].\mathrm{P}\rangle \xrightarrow{\overline{x}[\overrightarrow{K}]} \langle \Theta \cup \overrightarrow{\Gamma'}, \Gamma, \mathrm{P}\rangle}\ (out.)$$

$$\frac{\overrightarrow{\Theta'} \vdash_\pi \overrightarrow{K} : \overrightarrow{\sigma} \quad \bigcup \overrightarrow{\Theta'} \subseteq \Theta \quad \mathrm{dom} \bigcup \overrightarrow{\Theta'} = FN\left(\overrightarrow{K}\right)}{\langle \Theta, \Gamma, x(\overrightarrow{V} : \overrightarrow{\sigma}).\mathrm{P}\rangle \xrightarrow{x(\overrightarrow{K})} \langle \Theta, \Gamma \cup \overrightarrow{\Theta'}, \mathrm{P}\{\overrightarrow{K}/\overrightarrow{V}\}\rangle}\ (inp.)$$

$$\frac{\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle}{\langle \Theta, \Gamma, \mathrm{P} + \mathrm{Q}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle}\ (sum.)$$

$$\frac{\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle \quad BN(\mu) \cap FN(\mathrm{Q}) = \emptyset}{\langle \Theta, \Gamma, \mathrm{P}|\mathrm{Q}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'|\mathrm{Q}\rangle}\ (comp.)$$

$$\frac{\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle}{\langle \Theta, \Gamma, !\mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'|!\mathrm{P}\rangle}\ (rep.)$$

$$\frac{\langle \Theta, \Gamma_x \cup \{x : \sigma\}, \mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma'_x \cup \{x : \sigma\}, \mathrm{P}'\rangle \quad x \notin FN(\mu) \cup BN(\mu)}{\langle \Theta_x, \Gamma_x, (\nu x : \sigma)\mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta'_x, \Gamma'_x, (\nu x : \sigma)\mathrm{P}'\rangle}\ (res.)$$

$$\frac{\begin{array}{l}\langle \Theta, \Gamma_w \cup \{w : \sigma\}, \mathrm{P}\rangle \xrightarrow{(\nu \overrightarrow{z})\overline{x}[\overrightarrow{K}]} \langle \Theta', \Gamma'_w \cup \{w : \sigma\}, \mathrm{P}'\rangle \\ w \neq x,\ w \in FN\left(\overrightarrow{K}\right) - \overrightarrow{z}\end{array}}{\begin{array}{l}\langle \Theta_w, \Gamma_w, (\nu w : \sigma)\mathrm{P}\rangle \xrightarrow{(\nu w \overrightarrow{z})\overline{x}[\overrightarrow{K}]} \langle \Theta'_w, \Gamma'_w, (\nu w : \sigma)\mathrm{P}'\rangle\end{array}}\ (open)$$

$$\frac{\begin{array}{l}\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{(\nu \overrightarrow{z})\overline{x}[\overrightarrow{K}]} \langle \Theta', \Gamma', \mathrm{P}'\rangle \\ \langle \Gamma, \Theta, \mathrm{Q}\rangle \xrightarrow{x(\overrightarrow{K})} \langle \Gamma', \Theta', \mathrm{Q}\rangle \qquad \overrightarrow{z} \notin FN(\mathrm{Q})\end{array}}{\langle \Delta, \Gamma \cup \Theta, \mathrm{P}|\mathrm{Q}\rangle \xrightarrow{\tau} \langle \Delta, \Gamma' \cup \Theta', \mathrm{P}'|\mathrm{Q}'\rangle}\ (comm.)$$

Figure 2.7: Typed Labelled-Transition-Semantics for the Higher-order π-calculus

rather than type-based approach to enforce security properties (Hepburn and Wright 2001).

2.4 Discussion

This Chapter presented the basic — notation, semantics, and basic type systems — theory of the π-calculus, for both its first-order and higher-order versions. An explicitly-typed language was used in each case. The novel work to follow will all build on the notation and results presented in this chapter.

Chapter 3

Trust Type Systems

This chapter introduces the core novel ideas: the type systems and extension that implement the security model. There are two components to the systems: an implicit extension, visible only in the new type rules; and an explicit extension, a new syntactic construct.

The chapter commences with a presentation of the system of type annotations used, then in Section 3.2 the syntactic extension common to all variants of the system is introduced, with informal justifications and examples. Section 3.3 introduces the first of the type systems with a discussion of the first-order system. This serves as a basis for the remaining systems and is a powerful environment in its own right. In Section 3.4 these ideas are expanded upon by showing that many programs with strong intuitions behind them are none-the-less untypeable in the vanilla first-order system, and present a solution: sub-typing. This system in turn is completed in Section 3.5, with a minor addition to provide security by guarantees of non-interference from untrusted channels. A different notion of security is discussed and implemented for higher-order systems in Section 3.6. The focus in this system is on restricting the access of certain mobile agents to the host, according to some pre-determined security policy. This is a non-trivial extension of the first-order work, as it by necessity introduces some new notions about trust, and requires some novel approaches in order to accommodate them.

Note that there is no discussion of type safety, security properties, or implementations in this chapter; these are examined in Chapters 4 and 5 for safety/security and implementations respectively.

3.1 A System Of Type Annotations

Before introducing the systems it is first necessary to cover the form of annotations used. The annotations are modelled on a *Boolean algebra* (Boole 1847); this is not an original idea and follows on from similar work in (Wright 1992) and (Wright 1996). Using an algebra enables a variety of properties, conditionals and constraints to be encoded quite elegantly, as well as instantly providing a large body of previous results that may be leveraged in the analysis.

There are two extremes that need to be represented; trustedness (that is, data whose integrity can be verified), and untrustedness (data that has become corrupted, whether maliciously or accidentally). It seems natural to model these as truth and falsehood respectively (usually written as 1 and 0 in a boolean algebra). Rather than adopting these notational conventions, for clarity (a lower representational gap) in the work here T will represent trustedness and U untrustedness. Variables are also admitted in annotations; as well as permitting a smooth transition to type inference (see Chapter 5) where annotations for which concrete values are unable to be deduced may be assigned a variable, informally, and just as importantly, in the work that follows they also represent the situation in which a value is *unknown at compile time*. This situation is in fact quite common; any program that deals with data from an untrusted source which has an associated unforgeable digital signature is processing a value that may be either trustworthy or corrupted: the signature provides an avenue to determine this at *run-time* but its value at compile time cannot be ascertained with any certainty.

Definition 3.1.1 (Annotations) *Let* $\mathcal{B} = \{\mathrm{T}, \mathrm{U}\}$ *be the set of constant annotations, and* $\mathcal{B}_\mathcal{V}$ *the set of annotation variables ranged over by* i, j, k, *and* l. *Then* $\langle \mathcal{B} \cup \mathcal{B}_\mathcal{V}, +, \cdot, \bar{\ } \rangle$ *forms a boolean algebra. The set* $\mathcal{B}_\mathcal{A}$ *contains all terms in the algebra, and is ranged over by* b, c, d, *and* e. *With* T *corresponding to 1 and* U *to 0, the operations* $+$, $\cdot$ *and* $\bar{\ }$ *are the least operations satisfying the following:*

$$\begin{array}{cc} b \cdot \mathrm{T} = b & b \cdot \mathrm{U} = \mathrm{U} \\ b + \mathrm{T} = \mathrm{T} & b + \mathrm{U} = b \\ \overline{\mathrm{T}} = \mathrm{U} & \overline{\mathrm{U}} = \mathrm{T} \\ \overline{b} + b = \mathrm{T} & \overline{b} \cdot b = \mathrm{U} \end{array}$$

$$\overline{\overline{b}} = b$$

In the above, $+$ *and* $\cdot$ *are reflexive, transitive, distributive, and idempotent, and obey the usual laws of logical equivalence. Where it is clear the* $\cdot$ *operator*

may be omitted; for example $b \cdot c$ written just as bc to avoid visual clutter (particularly when used in superscripts).

Now the new type syntax may be presented (Definition 3.1.2).

Definition 3.1.2 (Annotated Type Syntax)

1. *The set of all types is denoted by $\mathcal{T}$, and ranged over by σ.*

2. *The new type syntax is exactly the same as in Definition 2.3.3, with the addition of an annotation (Definition 3.1.1 above) to each element:*

$$\sigma \quad ::= \quad (\overrightarrow{\sigma})^b$$

   Since the annotation will usually be of interest, the convention is adopted that σ^b may be written where b is the outermost annotation. This convention will in fact be used almost exclusively. It is similar to the presentation used in Wright (1996).

   Note that $\overrightarrow{\sigma}$ refers to the sequence $\sigma_1 \ldots \sigma_n$ for some finite n, and may be zero-length.

3. *Multiplication of types by variables is defined as a requirement on their structure: $(b \cdot \overrightarrow{\sigma^c})^b$ means that for each σ^{c_i} in the sequence $\overrightarrow{\sigma^c}$, $c_i = b \cdot c_i$ (that is, if $b = \mathrm{T}$ then c_i may be any arbitrary annotation; if $b = \mathrm{U}$ then c_i* must *be* U*).*

3.2 A Syntactic Extension For Safe Run-Time Coercion

The intuition behind the need for a syntactic extension can be seen by once again considering Examples 1.3.1 and 1.3.2. This example reflects a common desire in programming such systems; to cast a given input as either trusted or untrusted and act accordingly, based on the results of a known run-time verification procedure. This was of course the issue previously established; not only is this a common operation, but it is also error-prone when using common programming constructs (such as *if ... then ... else*) as most type-based safety checks are forced to assume that the programmer has correctly used such operations (since there is *no connection between the verification results and the subsequent branching*).

The novel addition to the π-calculus combines the operations of certification and branching (based on the results of the certification procedure used).

This simple combination turns out to have profoundly powerful implications: programmers may now continue programming unrestrained, and subject reduction holds so typed programs are guaranteed to be well-formed (including with respect to the integrity of data). The second consequence is they may now also use coercion of data integrity, but in an *implicit* manner: instead of explicitly casting a variable as either trusted or untrusted (then presumably branching to handle errors or continue with the secure calculation, for example) they now use a self-contained certify-and-branch construct and are guaranteed that one branch will be taken if the variable certifies (and hence may be considered trusted within the branch), and the other if certification fails (and thus the other branch always deals with an untrusted input).

Definition 3.2.1 (Oracles) *The integrity checks present in the system are represented as oracles. An oracle is a total function computing the integrity — trusted or untrusted — of channels. The metavariable certify will usually be used to range over oracles.*

The following judgement form specifies that certify is an oracle:

$$\vdash \text{certify} : \mathit{Oracle}$$

for some $\text{certify} \in \mathcal{N} \times \mathcal{B}$. *Function application is used to specify the result of an oracle, for example* $\text{certify}(x) = \text{T}$ *means that the oracle* certify *determined that the name* x *was trusted.*

The mechanism employed by the oracle is unspecified. The systems presented here simply assume that it is a function in the form above which is assumed to be correct. The judgement form used is intended to allow an extension of the system to use multiple different varieties of oracles, and to have their usage checked verified by the type-checker (that is, some oracles may only be partial functions whose usage must be restricted to certain categories of names). Chapter 6 elaborates slightly on this idea. The work here however assumes a single all-encompassing oracle, and the judgement form above is assumed to be implicit.

Following this informal justification, the syntactical and semantic implementation within the π-calculus may now be introduced. The syntax chosen is deliberately modelled on a similar construct in the C language (although the ":" used to separate the branches is replaced with the symbol $\oplus$, in order to avoid confusion with a similar symbol used in typing judgements), and is shown in Definition 3.2.2. Note that this is for the first-order case only; the syntax is identical for the higher-order calculus but is repeated for clarity later.

Definition 3.2.2 *The syntax is identical to that of Definition 2.2.2, with the addition of the new construct (shown on a new line for clarity):*

$$\begin{aligned} \mathrm{P} \quad ::= \quad & \mathbf{0} \mid \mathrm{P}|\mathrm{P} \mid \sum_{i\in I} \pi_i.\mathrm{P}_i \mid !\mathrm{P} \mid (\nu x : \sigma^b)\mathrm{P} \mid x[\overrightarrow{y}].\mathrm{P} \mid x(\overrightarrow{y} : \overrightarrow{\sigma^b}).\mathrm{P} \\ & \mid x(y : \sigma^i)\,{}_{\mathrm{certify}}\mathbf{?}\,\mathrm{P} \oplus \mathrm{P} \\ \pi \quad ::= \quad & x(\overrightarrow{y} : \overrightarrow{\sigma^b}) \mid \overline{x}[\overrightarrow{y}] \end{aligned}$$

Note that the new form is also guarded and thus can appear in a summation. Note also that the annotation must be a variable, as coercion only applies to variables. The subscript certify *specifies the oracle used (see Definition 3.2.1).*

The semantics will be formally specified shortly; informally though it resembles an input operation in that a (single) name is received along the channel x and bound to the name y; y is bound in both branches (the two processes P). Inside the first branch y is typed as trusted; inside the second as untrusted. This is what allows a compile-time guarantee of safety, modulo a sound certification operator: the compiler is now aware of the consequences of each branch, even if the programmer confuses the two, so any such (dangerous) mistake will be detected as a type error.

Definitions of substitution and free and bound names also need to be extended:

Definition 3.2.3 (Substitution in First-Order Certify Terms) *For the syntax of Definition 3.2.2, extend the definition of substitution of names (Definition 2.2.6, Page 21) to include*

$$(z(z' : \sigma^b)\,{}_{\mathrm{certify}}\mathbf{?}\,\mathrm{P} \oplus \mathrm{Q})\{y/x\} \quad \triangleq \quad z\{y/x\}(z' : \sigma^b)\,{}_{\mathrm{certify}}\mathbf{?}\,\mathrm{P}\{y/x\} \oplus \mathrm{Q}\{y/x\}$$

Definition 3.2.4 *For the syntax of Definition 3.2.2, extend the definition of free names (and by extension, bound names) in Definition 2.2.3 to include*

$$FN\big(x(y : \sigma^b)\,{}_{\mathrm{certify}}\mathbf{?}\,\mathrm{P} \oplus \mathrm{Q}\big) \quad \triangleq \quad (\{x\} \cup FN(\mathrm{P}) \cup FN(\mathrm{Q})) - \{y\}$$

3.2.1 Semantics of Certify

The semantics of the certify construct may also be presented in two fashions: as a rewriting rule, and as a labelled transition. Both must be parameterised on the oracle certify. A more precise specification may be defined using the typed labelled transition rules, and these will be presented next.

$$\frac{\text{certify}(y) = \text{T}}{x(z : \sigma^i)_{\text{certify}}?\text{P} \oplus \text{Q} \xrightarrow{x(y)} \text{P}\{y/z\}} \qquad \frac{\text{certify}(y) = \text{T}}{x(z : \sigma^i)_{\text{certify}}?\text{P} \oplus \text{Q} \xrightarrow{x(y)} \text{Q}\{y/z\}}$$

Figure 3.1: Certify Semantics; Labelled Transition Rules

Secondly, the reduction semantics (that is, as a rewrite rule) for completeness are also presented. The new rules here are essentially the same as the communication rule, however importantly they cannot be presented as axioms; there is a discriminated choice involved, and the choice depends on certify:

- Verification as trusted:

$$\frac{\text{certify}(y) = \text{T}}{\ldots + \overline{x}[y].\text{R}|x(z : \sigma^i)_{\text{certify}}?\text{P} \oplus \text{Q} + \ldots \rightarrow \text{R}|\text{P}\{y/z\}}$$

- Verification as untrusted:

$$\frac{\text{certify}(y) = \text{U}}{\ldots + \overline{x}[y].\text{R}|x(z : \sigma^i)_{\text{certify}}?\text{P} \oplus \text{Q} + \ldots \rightarrow \text{R}|\text{Q}\{y/z\}}$$

3.3 First-Order Type System

Several strong intuitions are carried into the design of the first-order type system:

- all data transmitted along an untrusted channel should itself be considered untrusted (as it may have been tampered with in transit);
- trusted channels however may be used to carry untrusted data (it will not be tampered with in transit, although it remains untrusted);
- some data effectively has unknown trustedness; there is perhaps potential for it to be corrupted, but this is detectable *at run-time* (for example, via secure digital signatures).

The rules implementing these intuitions are referred to collectively as System $\mathcal{T}_{\mathsf{FO}\pi}$, or occasionally just $\mathcal{T}_{\mathsf{FO}\pi}$. The key realisation is that the first two intuitions above can be enforced by imposing a simple constraint on all types allowed in the system, in the form of a multiplication.

Both intuitions, that data transmitted on an untrusted channel must be untrusted but that a trusted channel may carry untrusted data, are enforced if the types carried by the channel are multiplied by the trustedness of the channel. For example, if the carrying channel is untrusted and the type of the data carried has annotation $\mathrm{U} \cdot b$, then for any b the data carried must also be untrusted due to the rules of Definition 3.1.1. Similarly since T is the multiplicative identity in the same rules, if the host channel is trusted then $\mathrm{T} \cdot b = b$ for all b, including $b = \mathrm{U}$ as required.

The third intuition will be satisfied by the type rules and the use of the syntactic extension incorporating an oracle defined in Definitions 3.2.1 and 3.2.2.

For differentiation the rules will be presented using different subscripts in the judgements, as defined in Definition 3.3.1:

Definition 3.3.1 (Judgements for System $\mathcal{T}_{\mathsf{FO}\pi}$) *The judgements in System $\mathcal{T}_{\mathsf{FO}\pi}$ will take the two forms shown below:*

$$\Gamma \quad \vdash_{\mathrm{FO}} \quad \mathrm{P} : \mathsf{Proc}$$
$$\Gamma \quad \vdash_{(\mathrm{FO})} \quad \mathrm{P} : \mathsf{Proc}$$

These are as in Definition 2.3.2, with the use of a different subscript to differentiate the type rules in operation. As before, the parenthesised subscript form is only used for null, guarded, or summation processes.

A third judgement form will be used to specify that a particular type *is well-formed under the same rules:*

$$\vdash_{\mathrm{FO}} \sigma^b$$

The complete set of rules are shown in Figure 3.2; the pertinent cases and justification behind them will now be examined. The rules for replication, restriction, summation, and composition are unchanged from the base case (barring the addition of annotations to the types where necessary).

The first departure from the unadorned rules is the inclusion of rules for weakening and well-formed types in System $\mathcal{T}_{\mathsf{FO}\pi}$. This is to guarantee, via an antecedent in the weaken rule, that all types be well-formed according to the rule *type*. The rule for well-formed types itself enforces the intuitions discussed early concerning the relationship between the trustedness of data carried by a channel, and the trustedness of the channel itself. To complement this strategy, the rule *zero* states that it is initially well-formed under the empty environment, instead of an arbitrary environment. Note that there is no corresponding contraction rule, as the second antecedent to the weaken

$$\frac{\vdash_{\mathrm{FO}} \overrightarrow{\sigma^b}}{\vdash_{\mathrm{FO}} (c \cdot \overrightarrow{\sigma^b})^c}\ (type.) \qquad \frac{\Gamma \vdash_{(\mathrm{FO})} \mathrm{P} : \mathsf{Proc}}{\Gamma \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}}\ (conv.)$$

$$\frac{}{\emptyset \vdash_{(\mathrm{FO})} \mathbf{0} : \mathsf{Proc}}\ (zero.) \qquad \frac{\Gamma \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc} \quad x \notin \mathrm{dom}\Gamma \quad \vdash \sigma^b}{\Gamma, x : \sigma^b \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}}\ (weak.)$$

$$\frac{\Gamma \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}}{\Gamma \vdash_{\mathrm{FO}} !\mathrm{P} : \mathsf{Proc}}\ (rep.) \qquad \frac{\Gamma \vdash_{(\mathrm{FO})} \mathrm{P} : \mathsf{Proc} \quad \Theta \vdash_{(\mathrm{FO})} \mathrm{Q} : \mathsf{Proc}}{\Gamma, \Theta \vdash_{(\mathrm{FO})} \mathrm{P} + \mathrm{Q} : \mathsf{Proc}}\ (sum.)$$

$$\frac{\Gamma_x, x : \sigma^b \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}}{\Gamma_x \vdash_{\mathrm{FO}} (\nu x : \sigma^b)\mathrm{P} : \mathsf{Proc}}\ (res.) \qquad \frac{\Gamma \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc} \quad \Theta \vdash_{\mathrm{FO}} \mathrm{Q} : \mathsf{Proc}}{\Gamma, \Theta \vdash_{\mathrm{FO}} \mathrm{P}|\mathrm{Q} : \mathsf{Proc}}\ (comp.)$$

$$\frac{\Gamma, x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma^b} \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}}{\Gamma, x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma^b} \vdash_{(\mathrm{FO})} \overline{x}[\overrightarrow{y}].\mathrm{P} : \mathsf{Proc}}\ (out.)$$

$$\frac{\Gamma_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma^b} \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}}{\Gamma_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^b})^c \vdash_{(\mathrm{FO})} x(\overrightarrow{y} : \overrightarrow{\sigma^b}).\mathrm{P} : \mathsf{Proc}}\ (inp.)$$

$$\frac{\Gamma_y, x : (\sigma^i)^b, y : \sigma^{\mathrm{T}} \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc} \quad \Theta_y, x : (\sigma^i)^b, y : \sigma^{\mathrm{U}} \vdash_{\mathrm{FO}} \mathrm{Q} : \mathsf{Proc}}{\Gamma_y, \Theta_y, x : (\sigma^i)^b \vdash_{(\mathrm{FO})} x(y : \sigma^i)_{\underset{\text{certify}}{?}}\mathrm{P} \oplus \mathrm{Q} : \mathsf{Proc}}\ (cert.)$$

Figure 3.2: Type Rules Of System $\mathcal{T}_{\mathsf{FO}\pi}$

rule requires that the name being introduced does not already exist in the current environment.

The other interesting cases are those dealing with input/output and certification; these will now be examined individually.

Input:

$$\frac{\Gamma_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma^b} \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}}{\Gamma_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^b})^c \vdash_{(\mathrm{FO})} x(\overrightarrow{y} : \overrightarrow{\sigma^b}).\mathrm{P} : \mathsf{Proc}}$$

This rule has many similarities to the base case: the types of the arguments ($\overrightarrow{y}$) must match the types carried by the channel x; and it is a binding rule so the environment in the antecedent has had the names $\overrightarrow{y}$ removed from its domain.

The most significant departure from the base case (see Figure 2.4, Page 36) is in the annotations and their implications. The well-formed type rule guarantees that types of the *operands* of x are multiplied by the annotation *of* x. That is, if $\overrightarrow{\sigma^b}$ represents $\sigma_1^{b_1} \ldots \sigma_n^{b_n}$ then each b_i must be equivalent to $c \cdot b_i$.

This is at once very simple, and very powerful; it concisely encapsulates the first two intuitions above. To see how, it is worth while to consider a few examples. If x is a *trusted* channel, then $c = \mathrm{T}$ (in the rule above), and since T is the *multiplicative identity* then *any* annotation is allowed in the operand of x (since $b = \mathrm{T} \cdot b$ for any b). In other words a trusted channel may be used to carry names of any level of trustedness, including untrusted, as per the intuitive requirement.

Alternatively, if x is an *untrusted* channel (that is, in the rule above $c = \mathrm{U}$) there is a different scenario. Because U is the *zero* multiplicative element in the algebra (meaning that for all b then $\mathrm{U} \cdot b = \mathrm{U}$) then by definition x may only carry other untrusted names, since $\mathrm{U} \cdot \sigma_1^{b_1} \ldots \mathrm{U} \cdot \sigma_n^{b_n}$ implies that all b_i are identical to U. This too captures the intuition that all data passing through an untrusted channel must also be considered untrustworthy, due to the risk of corruption.[1]

Output: The rule

$$\frac{\Gamma, x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma^b} \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}}{\Gamma, x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma^b} \vdash_{(\mathrm{FO})} \overline{x}[\overrightarrow{y}].\mathrm{P} : \mathsf{Proc}}$$

is similar to the input rule (although more divergences will be noted in the other systems; see Sections 3.4 and 3.6), with the exception that since it is

[1] An alternative perspective would be to allow trusted values to sent along untrusted channels, but be untrusted at the receiving end (thus potentially allowing more programs to be typed). This avenue is explored in Section 3.4.

not a binding rule on its operands the environment is unchanged from the antecedent to the consequent.

Certify: The rule

$$\frac{\Gamma_y, x : (\sigma^i)^b, y : \sigma^{\mathrm{T}} \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc} \quad \Theta_y, x : (\sigma^i)^b, y : \sigma^{\mathrm{U}} \vdash_{\mathrm{FO}} \mathrm{Q} : \mathsf{Proc}}{\Gamma_y, \Theta_y, x : (\sigma^i)^b \vdash_{(\mathrm{FO})} x(y : \sigma^i)_{\mathrm{certify}}^{?} \mathrm{P} \oplus \mathrm{Q} : \mathsf{Proc}}$$

is the cornerstone of the system. Each antecedent is at first glance much like the construction of the input rule, and indeed it is a form of input; the operand is bound and therefore also discharged from the environment in the consequent. The first differences become apparent when considering the operand (that is, the variable whose integrity will — usually — be determined at run-time) in each antecedent: in one branch it is typed as trusted ($y : \sigma^{\mathrm{T}}$), and in the other as untrusted. The input channel itself is typed as carrying a single name with a variable annotation (note that this implies $i = b \cdot i$ due to the well-formed type rule). This is where coercion fits in: a value of variable trustedness — that is, one whose integrity can only be determined at run-time — is received, and this variable is then coerced at run-time based on the results of the certify oracle. This is what gives the rule its power: an input form is constructed as per normal, but can then *safely* assume (since the entire branch, either P or Q, is typed on that assumption) that the name received is trusted in one branch and untrusted in the other. In other words, if the programmer now confuses the branches this will be detected by the *type* system. For example the second branch must be parameterised on an untrusted variable so will cause a type-check error if such an input is used in a trusted context.

3.3.1 Typed Labelled Transition Semantics

The typed semantics provide additional insight in to the nature of the coercion property of certify. They are mostly similar — apart from the addition of annotations to the types — from those for the plain first-order calculus in Figure 2.5 (Page 37).

The most obvious change is of course in the addition of rules for certify. The possibility of coercion during a reduction, due to certify, does change things slightly however. Because the effect of a coercion should be global, the structural rules for combining processes need to propagate the possible coercion to the new processes.

Coercions are represented by substitutions on annotations variables (Definition 3.3.2):

Definition 3.3.2 (Annotation Substitutions) *An annotation substitution is a function ranged over by $\mathbb{R}$ from annotation variables ($\mathcal{B}_\mathcal{V}$) to annotation terms ($\mathcal{B}_\mathcal{A}$). The identity substitution is written as Id, and $\mathbb{R}[i := b]$ represents the substitution identical to $\mathbb{R}$, extended to substitute the term b for the variable i.*

Composition of substitutions is written in-order as $\mathbb{R}_1;\mathbb{R}_2$, in place of the more familiar $\mathbb{R}_2 \circ \mathbb{R}_1$, so $\mathbb{R}_1;\mathbb{R}_2(b)$ means $\mathbb{R}_2(\mathbb{R}_1(b))$.

Substitutions can be applied to other objects in the logical way:

- $\mathbb{R}((\overrightarrow{\sigma^c})^b) \triangleq (\mathbb{R}(\overrightarrow{\sigma^c}))^{\mathbb{R}(b)}$
- $\mathbb{R}(\Gamma) \triangleq \{x : \mathbb{R}(\sigma^b) | x : \sigma^b \in \Gamma\}$
- *Because the language is typed, terms in the syntax may contain annotation variables too:*

$$\begin{array}{rcl}
\mathbb{R}(\mathbf{0}) & \triangleq & \mathbf{0} \\
\mathbb{R}(!\mathrm{P}) & \triangleq & !\mathbb{R}(\mathrm{P}) \\
\mathbb{R}((\nu x : \sigma^b)\mathrm{P}) & \triangleq & (\nu x : \mathbb{R}(\sigma^b))\mathbb{R}(\mathrm{P}) \\
\mathbb{R}(\mathrm{P}|\mathrm{Q}) & \triangleq & \mathbb{R}(\mathrm{P}) \mid \mathbb{R}(\mathrm{Q}) \\
\mathbb{R}(\mathrm{P} + \mathrm{Q}) & \triangleq & \mathbb{R}(\mathrm{P}) + \mathbb{R}(\mathrm{Q}) \\
\mathbb{R}(\overline{x}[\overrightarrow{y}].\mathrm{P}) & \triangleq & \overline{x}[\overrightarrow{y}].\mathbb{R}(\mathrm{P}) \\
\mathbb{R}(x(\overrightarrow{y} : \overrightarrow{\sigma^b}).\mathrm{P}) & \triangleq & x(\overrightarrow{y} : \mathbb{R}(\overrightarrow{\sigma^b})).\mathbb{R}(\mathrm{P}) \\
\mathbb{R}(x(y : \sigma^i)\,\underset{\text{certify}}{?}\,\mathrm{P} \oplus \mathrm{Q}) & \triangleq & x(y : \mathbb{R}(\sigma^i))\,\underset{\text{certify}}{?}\,\mathbb{R}(\mathrm{P}) \oplus (\mathrm{Q})
\end{array}$$

As an abbreviation, substitutions will sometimes be applied to entire judgements as well, for example

$$\mathbb{R}(\Gamma \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc})$$

being used as a short-hand for

$$\mathbb{R}(\Gamma) \vdash_{\mathrm{FO}} \mathbb{R}(\mathrm{P}) : \mathsf{Proc}$$

(this becomes a greater convenience for the more heavily adorned judgements encountered later on).

Now, the syntax of the typed labelled transition rules is extended to capture the coercions that occur:

$$\langle \Theta, \Gamma, \mathrm{P} \rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}' \rangle \Vdash_{\text{certify}} \mathbb{R}$$

As before, this means that a process P, typed under environment Γ, may perform the action μ, possibly by interacting with an anonymous process

typed under environment Θ (where $\Gamma \asymp \Theta$) to transform itself to P' and the two environments to Γ' and Θ'. Additionally however the substitution $\mathbb{R}$ represents precisely all coercions due to certify that occurred during the reduction.

There are two new cases in the new rules, presented in Figure 3.3, one each for coercion as trusted and untrusted. Note the application of the substitution to the other term in the cases for composition and replication, as well as of course in the cases for certify. The multi-step closure is written as

$$\langle \Theta, \Gamma, \mathrm{P} \rangle \xrightarrow{\vec{\mu}} \langle \Theta', \Gamma', \mathrm{P}' \rangle \Vdash_{\text{certify}} \mathbb{R}$$

where

$$\frac{\langle \Theta, \Gamma, \mathrm{P} \rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}' \rangle \Vdash_{\text{certify}} \mathbb{R}_1 \quad \langle \Theta', \Gamma', \mathrm{P}' \rangle \xrightarrow{\vec{\mu}} \langle \Theta'', \Gamma'', \mathrm{P}'' \rangle \Vdash_{\text{certify}} \mathbb{R}_2}{\langle \Theta, \Gamma, \mathrm{P} \rangle \xrightarrow{\mu \vec{\mu}} \langle \Theta'', \Gamma'', \mathrm{P}'' \rangle \Vdash_{\text{certify}} \mathbb{R}_1; \mathbb{R}_2}$$

3.4 First-Order System With Sub-typing

As powerful as the system in the previous section is, there remain many programs that cannot be typed under it.

Consider the following:

$$\Gamma, \mathit{noise} : (\sigma^i)^i, \mathit{data} : \sigma^{\mathrm{T}} \vdash_{\mathrm{FO}} \overline{\mathit{noise}}[\mathit{data}].\mathbf{0} \mid \mathit{noise}(y : \sigma^i)_{\text{certify}}^{?} \mathit{handleData} \oplus \mathit{Error}$$

In this situation, trusted data is sent along a noisy channel (that is, it may become corrupted), and at the receiving end some verification process occurs — perhaps simply checksum verification — and then either branches to some error-handling process or proceeds as anticipated. None the less, as the data is declared as being initially trusted this program would be rejected under the rules of System $\mathcal{T}_{\mathsf{FO}\pi}$. This is a particularly egregious situation given that a verification procedure exists at the receiving end. In addition, given that there is no loss of safety — the receiver will always treat it as having variable trustedness i — it would seem a valid program anyway.

As a second example, consider the following conservative program:

$$\Gamma, \mathit{sandbox} : \sigma^{\mathrm{U}}, x : (\sigma^{\mathrm{T}})^{\mathrm{T}} \vdash_{\mathrm{FO}} \mathit{paranoid} : \mathsf{Proc}$$

The intent of this particular program is to place all data — no matter what integrity level — in an untrusted *sandbox* variable to minimise its danger.

$$\frac{\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle \Vdash_{\text{certify}} \mathbb{R} \quad \mathrm{P} \equiv_\alpha \mathrm{Q}}{\langle \Theta, \Gamma, \mathrm{Q}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle \Vdash_{\text{certify}} \mathbb{R}} \ (alp.)$$

$$\frac{}{\langle \Theta, \Gamma \cup \{\overrightarrow{y} : \overrightarrow{\sigma^b}\}, \overline{x}[\overrightarrow{y}].\mathrm{P}\rangle \xrightarrow{\overline{x}[\overrightarrow{y}]} \langle \Theta \cup \{\overrightarrow{y} : \overrightarrow{\sigma^b}\}, \Gamma \cup \{\overrightarrow{y} : \overrightarrow{\sigma^b}\}, \mathrm{P}\rangle \Vdash_{\text{certify}} \mathsf{Id}} \ (out.)$$

$$\frac{}{\langle \Theta \cup \{\overrightarrow{y} : \overrightarrow{\sigma^b}\}, \Gamma, x(\overrightarrow{z} : \overrightarrow{\sigma^b}).\mathrm{P}\rangle \xrightarrow{x(\overrightarrow{y})} \langle \Theta \cup \{\overrightarrow{y} : \overrightarrow{\sigma^b}\}, \Gamma \cup \{\overrightarrow{y} : \overrightarrow{\sigma^b}\}, \mathrm{P}\{\overrightarrow{y}/\overrightarrow{z}\}\rangle \Vdash_{\text{certify}} \mathsf{Id}} \ (inp.)$$

$$\frac{\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle \Vdash_{\text{certify}} \mathbb{R}}{\langle \Theta, \Gamma, \mathrm{P} + \mathrm{Q}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle \Vdash_{\text{certify}} \mathbb{R}} \ (sum.)$$

$$\frac{\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle \Vdash_{\text{certify}} \mathbb{R} \quad BN(\mu) \cap FN(\mathrm{Q}) = \emptyset}{\langle \Theta, \Gamma, \mathrm{P}|\mathrm{Q}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'|\mathbb{R}(\mathrm{Q})\rangle \Vdash_{\text{certify}} \mathbb{R}} \ (comp.)$$

$$\frac{\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle \Vdash_{\text{certify}} \mathbb{R}}{\langle \Theta, \Gamma, !\mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'|\mathbb{R}(!\mathrm{P})\rangle \Vdash_{\text{certify}} \mathbb{R}} \ (rep.)$$

$$\frac{\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle \Vdash_{\text{certify}} \mathbb{R} \quad x \notin FN(\mu) \cup BN(\mu)}{\langle \Theta_x, \Gamma_x, (\nu x)\mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta'_x, \Gamma'_x, (\nu x)\mathrm{P}'\rangle \Vdash_{\text{certify}} \mathbb{R}} \ (res.)$$

$$\frac{\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{(\nu \overrightarrow{z})\overline{x}[\overrightarrow{y}]} \langle \Theta', \Gamma', \mathrm{P}'\rangle \Vdash_{\text{certify}} \mathsf{Id} \quad w \neq x,\ w \in \overrightarrow{y} - \overrightarrow{z}}{\langle \Theta_w, \Gamma_w, (\nu w)\mathrm{P}\rangle \xrightarrow{(\nu w \overrightarrow{z})\overline{x}[\overrightarrow{y}]} \langle \Theta'_w, \Gamma'_w, (\nu w)\mathrm{P}'\rangle \Vdash_{\text{certify}} \mathsf{Id}} \ (open)$$

$$\frac{\begin{array}{l}\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{(\nu \overrightarrow{z})\overline{x}[\overrightarrow{y}]} \langle \Theta', \Gamma', \mathrm{P}'\rangle \Vdash_{\text{certify}} \mathsf{Id} \\ \langle \Gamma, \Theta, \mathrm{Q}\rangle \xrightarrow{x(\overrightarrow{y})} \langle \Gamma', \Theta', \mathrm{Q}\rangle \Vdash_{\text{certify}} \mathbb{R} \qquad \overrightarrow{z} \notin FN(\mathrm{Q})\end{array}}{\langle \Delta, \Gamma \cup \Theta, \mathrm{P}|\mathrm{Q}\rangle \xrightarrow{\tau} \langle \mathbb{R}(\Delta), \mathbb{R}(\Gamma' \cup \Theta), \mathbb{R}(\mathrm{P}'|\mathrm{Q}')\rangle \Vdash_{\text{certify}} \mathbb{R}} \ (comm.)$$

$$\frac{\text{certify}(z) = \mathrm{T} \quad \mathbb{R} = \mathsf{Id}[i := \mathrm{T}]}{\langle \Theta, \Gamma \cup \{x : (\sigma^i)^b\}, x(y)_{\text{certify}}^{?}\mathrm{P} \oplus \mathrm{Q}\rangle \xrightarrow{x(z)} \langle \mathbb{R}(\Theta), \mathbb{R}(\Gamma \cup \{x : (\sigma^i)^b, z : \sigma^{\mathrm{T}}\}), \mathbb{R}(\mathrm{P}\{z/y\})\rangle} \ (cert - T)$$

$$\frac{\text{certify}(z) = \mathrm{U} \quad \mathbb{R} = \mathsf{Id}[i := \mathrm{U}]}{\langle \Theta, \Gamma \cup \{x : (\sigma^i)^b\}, x(y)_{\text{certify}}^{?}\mathrm{P} \oplus \mathrm{Q}\rangle \xrightarrow{x(z)} \langle \mathbb{R}(\Theta), \mathbb{R}(\Gamma \cup \{x : (\sigma^i)^b, z : \sigma^{\mathrm{T}}\}), \mathbb{R}(\mathrm{Q}\{z/y\})\rangle} \ (cert - U)$$

Figure 3.3: First-Order Typed Labelled Transition Semantics

However, with the rules of System $\mathcal{T}_{\mathsf{FO}\pi}$ it is not possible to construct the program

$$x(sandbox : \sigma^{\mathrm{U}}).paranoid$$

because the channel x is typed as carrying a trusted variable.

These examples may essentially be boiled down to another set of intuitions that a flexible system should allow, but that would not be permitted within the rules of Figure 3.2.

The primary assumption, or intuition, is that a name that is trusted may be treated as untrusted (that is, with caution!) with no loss of safety to the system. This leads to two consequences:

- it should be possible to bind trusted names to untrusted variables (but *not* vice versa); and
- it should be possible to send trusted names along an untrusted channel if so desired, although they will be (rightly) treated as untrusted at the destination.

An important implication of these points is that they essentially enable *local* trustedness; that is the trustedness of a variable may vary in the deduction tree, being considered trusted locally but untrusted globally for example. The rules implementing these assumptions — shown in Figure 3.4 — are collectively referred to as System $\mathcal{T}_{\mathsf{FO}\pi\leq}$ and require a notion of *sub-typing*. This concept is examined next in Section 3.4.1, followed by individual examination of the noteworthy rules. The judgement forms used are described in Definition

Definition 3.4.1 (Judgements of System $\mathcal{T}_{\mathsf{FO}\pi\leq}$) *The judgement forms of System $\mathcal{T}_{\mathsf{FO}\pi\leq}$ are identical to those of System $\mathcal{T}_{\mathsf{FO}\pi}$, with the exception of a different subscript for differentiation:*

$$\Gamma \quad \vdash_{\mathrm{FO}\leq} \quad \mathrm{P} : \mathsf{Proc}$$
$$\Gamma \quad \vdash_{(\mathrm{FO}\leq)} \quad \mathrm{P} : \mathsf{Proc}$$

have the usual meaning of well-formed process and well-formed null, guarded, or summation process as before. Also as before, the judgement

$$\vdash_{\mathrm{FO}\leq} \sigma^b$$

means that the type σ^b is well-formed under the rules of System $\mathcal{T}_{\mathsf{FO}\pi\leq}$.

$$\frac{\vdash_{\mathrm{FO}\leq} \overrightarrow{\sigma^b}}{\vdash_{\mathrm{FO}\leq} (c \cdot \overrightarrow{\sigma^b})^c}\ (type.) \qquad \frac{\Gamma \vdash_{(\mathrm{FO}\leq)} \mathrm{P} : \mathsf{Proc}}{\Gamma \vdash_{\mathrm{FO}\leq} \mathrm{P} : \mathsf{Proc}}\ (conv.)$$

$$\frac{}{\emptyset \vdash_{(\mathrm{FO}\leq)} \mathbf{0} : \mathsf{Proc}}\ (zero) \qquad \frac{\Gamma \vdash_{\mathrm{FO}\leq} \mathrm{P} : \mathsf{Proc} \quad x \notin \mathrm{dom}\Gamma \quad \vdash_{\mathrm{FO}\leq} \sigma^b}{\Gamma, x : \sigma^b \vdash_{\mathrm{FO}\leq} \mathrm{P} : \mathsf{Proc}}\ (weak.)$$

$$\frac{\Gamma \vdash_{\mathrm{FO}\leq} \mathrm{P} : \mathsf{Proc}}{\Gamma \vdash_{\mathrm{FO}\leq} !\mathrm{P} : \mathsf{Proc}}\ (rep.) \qquad \frac{\Gamma \vdash_{(\mathrm{FO}\leq)} \mathrm{P} : \mathsf{Proc} \quad \Theta \vdash_{(\mathrm{FO}\leq)} \mathrm{Q} : \mathsf{Proc}}{\Gamma, \Theta \vdash_{(\mathrm{FO}\leq)} \mathrm{P} + \mathrm{Q} : \mathsf{Proc}}\ (sum.)$$

$$\frac{\Gamma_x, x : \sigma^b \vdash_{\mathrm{FO}\leq} \mathrm{P} : \mathsf{Proc}}{\Gamma_x \vdash_{\mathrm{FO}\leq} (\nu x : \sigma^b)\mathrm{P} : \mathsf{Proc}}\ (res.) \qquad \frac{\Gamma \vdash_{\mathrm{FO}\leq} \mathrm{P} : \mathsf{Proc} \quad \Theta \vdash_{\mathrm{FO}\leq} \mathrm{Q} : \mathsf{Proc}}{\Gamma, \Theta \vdash_{\mathrm{FO}\leq} \mathrm{P}|\mathrm{Q} : \mathsf{Proc}}\ (comp.)$$

$$\frac{\Gamma_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma'^{b'}} \vdash_{\mathrm{FO}\leq} \mathrm{P} : \mathsf{Proc} \quad \overrightarrow{\sigma^b} \leq \overrightarrow{\sigma'^{b'}}}{\Gamma_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^b})^c \vdash_{(\mathrm{FO}\leq)} x(\overrightarrow{y} : \overrightarrow{\sigma^b}).\mathrm{P} : \mathsf{Proc}}\ (inp.)$$

$$\frac{\Gamma, x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma'^{b'}} \vdash_{\mathrm{FO}\leq} \mathrm{P} : \mathsf{Proc} \quad \overrightarrow{\sigma'^{b'}} \leq \overrightarrow{\sigma^b}}{\Gamma, x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma'^{b'}} \vdash_{(\mathrm{FO}\leq)} \overline{x}[\overrightarrow{y}].\mathrm{P} : \mathsf{Proc}}\ (out.)$$

$$\frac{\Gamma_y, x : (\sigma^i)^b, y : \sigma^{\mathrm{T}} \vdash_{\mathrm{FO}\leq} \mathrm{P} : \mathsf{Proc} \quad \Theta_y, x : (\sigma^i)^b, y : \sigma^{\mathrm{U}} \vdash_{\mathrm{FO}\leq} \mathrm{Q} : \mathsf{Proc}}{\Gamma_y, \Theta_y, x : (\sigma^c)^b \vdash_{(\mathrm{FO}\leq)} x(y : \sigma^i)_{\mathrm{certify}}?\mathrm{P} \oplus \mathrm{Q} : \mathsf{Proc}}\ (cert.)$$

Figure 3.4: Type Rules Of System $\mathcal{T}_{\mathsf{FO}\pi\leq}$

3.4.1 Subtyping

Informally, a type τ_1 is said to be a *sub-type* of another type τ_2 — usually written as $\tau_1 \leq \tau_2$ — if and only if in any situation that may be satisfied with an instance of type τ_2 an instance of τ_1 may be used instead with no loss of safety.

For example, a calculation involving real numbers may be satisfied using an integer value; integers are a *sub-type* of reals and every integer is also a real number. Note that the reverse is not true; not every real number is also an integer so it is not safe to use real numbers in an integer calculation.

In implementing sub-typing for the systems under consideration here an ordering is first defined on trust annotations (Definition 3.4.2):

Definition 3.4.2 (Trust Annotation Ordering) *From the boolean algebra* $\langle \mathcal{B} \cup \mathcal{B}_\mathcal{V}, +, \cdot, \bar{} \rangle$ *of annotations the corresponding lattice* $\langle \mathcal{B} \cup \mathcal{B}_\mathcal{V}, \leq_\mathsf{B} \rangle$ *is generated in the standard way, where* T *and* U *are the supremum and infinum respectively, the meet and join operators are* $\cdot$ *and* $+$*, and the ordering* $\leq_\mathsf{B}$ *is defined as*

$$b \leq_\mathsf{B} c \iff b \cdot c = b$$

Likewise, define the reverse operation $b \geq_\mathsf{B} c$ *iff* $c \leq_\mathsf{B} b$*.*

This definition however runs counter to the desired sub-typing assumption: a trusted name may be used in place of an untrusted variable, implying that a trusted type is a sub-type of an untrusted type — the *reverse* implied by the ordering of Definition 3.4.2. Accordingly, sub-typing is defined as follows (Definition 3.4.3):

Definition 3.4.3 *The relation* $\leq$ *on types is the least transitive and reflexive relation satisfying the following:*

$$\sigma_1^{b_1} \leq \sigma_2^{b_2} \iff \forall c. \sigma_1^c \equiv \sigma_2^c \wedge b_2 \leq_\mathsf{B} b_1$$

If $\sigma_1^{b_1} \leq \sigma_2^{b_2}$ *then* $\sigma_1^{b_1}$ *is a* sub-type *of* $\sigma_2^{b_2}$ *(Note the contravariant use of* $\leq_\mathsf{B}$ *on the annotations). Similarly, define the reverse operation* $\geq$ *as* $\sigma_2^{b_2} \geq \sigma_1^{b_1}$ *iff* $\sigma_1^{b_1} \leq \sigma_2^{b_2}$*.*

There is an important point to note here: the sub-typing is expressed *solely in terms of the outer-most annotations* (the base types must be congruent). This does lead to some loss of flexibility, however if the relation is to extend any deeper into the type structure it is necessary to include additional information about input/output usage in the type and thus the rules here are restricted to the surface case. Section 6.1.1 has more discussion on this point, with suggestions on how future work in this direction might proceed.

To understand the reasons behind the need for usage information, consider for a moment the well-known function-subtyping rule (Cardelli 1996). To review, a function with type $\rho_1 \to \tau_1$ is a sub-type of the function with type $\rho_2 \to \tau_2$ if $\tau_1 \leq \tau_2$ *and* $\rho_2 \leq \rho_1$: that is, if the first function returns results of type τ_1 then logically it must also return results of the supertype τ_2; and similarly if it accepts inputs of type ρ_1 it also accepts inputs of the sub-type ρ_2. Thus, the first function may be used in place of the second, and hence $\rho_1 \to \tau_1$ is a sub-type of $\rho_2 \to \tau_2$.

A similar situation exists with the nested structures of the π-calculus types. If a channel *outputs* an untrusted name, then intuitively one would expect to also be able to use a variable of a similar type that outputs a *trusted* name. The situation is reversed however when considering a name used for input; a name being used to bind a trusted value should also be able to bind untrusted values. Thus any sub-typing relation involving deeper consideration than the outer-most annotations also requires information about input/output usage of the channel (so that a full recursive comparison may be made), and no such avenues are pursued here. See Pierce and Sangiorgi (1993) for a complete coverage of input/output types and sub-typing, and Igarashi and Kobayashi (2000, Section 2.4) for a comparison of the two techniques (outermost annotation only versus a full structural comparison) with comprehensive discussion of the implications.

Interestingly, due to the fact that the systems here do *not* contain usage information, this notion of contravariant annotation orderings is expressed in the form of the rules themselves (see Figure 3.4)

The only alteration in the typed semantics required is in the input rule to accommodate sub-typing:

$$\frac{\overrightarrow{\sigma^b} \leq \overrightarrow{\sigma^c}}{\begin{array}{l}\langle \Theta \cup \{\overrightarrow{y} : \overrightarrow{\sigma^b}\}, \Gamma, x(\overrightarrow{z} : \overrightarrow{\sigma^c}).\mathrm{P}\rangle \xrightarrow{x(\overrightarrow{y})} \\ \qquad \langle \Theta \cup \{\overrightarrow{y} : \overrightarrow{\sigma^b}\}, \Gamma \cup \{\overrightarrow{y} : \overrightarrow{\sigma^b}\}, \mathrm{P}\{\overrightarrow{y}/\overrightarrow{z}\}\rangle \Vdash_{\text{certify}} \mathsf{Id}\end{array}}$$

No other change to the typed semantics from Figures 3.3 is required.

3.4.2 Analysis of the Rules

The rules well-formed types, conversion, zero, weaken, restriction, replication, summation, composition, and certify are unchanged from those for System $\mathcal{T}_{\mathsf{FO}\pi}$. Where things become interesting is in the examination of the interaction rules: input and output (the rule for certify does not change, as the annotation on the bound name is fixed by the rule).

Input: The rule

$$\frac{\Gamma_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma'^{b'}} \vdash_{\mathrm{FO}\leq} \mathrm{P} : \mathsf{Proc} \quad \overrightarrow{\sigma^b} \leq \overrightarrow{\sigma'^{b'}}}{\Gamma_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^b})^c \vdash_{(\mathrm{FO}\leq)} x(\overrightarrow{y} : \overrightarrow{\sigma^b}).\mathrm{P} : \mathsf{Proc}}$$

retains the expected similarities with the plain first-order system, but also subtly diverges. The bound names are removed from the environment in the consequent as would be expected, and the well-formed type rule guarantees that all types carried by the input channel x are multiplied by the trustedness of x, however the types of the operands $(\overrightarrow{y} : \sigma'^{b'})$ are slightly different. They are reconciled by the new clause $\overrightarrow{\sigma^b} \leq \overrightarrow{\sigma'^{b'}}$ in the antecedent. That is, the types x receives are a *sub-type* of the names being bound.

The implications of this are that while x may input trusted names (implying that x itself is trusted, as a consequence of the multiplication clause in its type), it may in fact bind these names to *untrusted* variables (recall Definition 3.4.3, noting the reverse ordering of the annotations). As was informally expressed at the beginning of Section 3.4 this is a sound construction, provided that the reverse situation can never occur, since no safety is lost: merely a trusted value treated as untrusted. This naturally allows more programs to be safely typed.

Output: The rule

$$\frac{\Gamma, x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma'^{b'}} \vdash_{\mathrm{FO}\leq} \mathrm{P} : \mathsf{Proc} \quad \overrightarrow{\sigma'^{b'}} \leq \overrightarrow{\sigma^b}}{\Gamma, x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma'^{b'}} \vdash_{(\mathrm{FO}\leq)} \overline{x}[\overrightarrow{y}].\mathrm{P} : \mathsf{Proc}}$$

is extended similarly. As could be expected the well-formed type rule enforces the policy that all arguments must be multiplied by the trustedness of x. Similarly to the input rule the types of the arguments themselves are not required to perfectly match the type of x, provided that they are related by the sub-typing clause $\overrightarrow{\sigma'^{b'}} \leq \overrightarrow{\sigma^b}$ (where x has type $(\overrightarrow{\sigma^b})^c$, and $\overrightarrow{y} : \overrightarrow{\sigma'^{b'}}$). Observe, however, that the order of the relation is the *reverse* of that in the input rule.

This has the natural implication (see Definition 3.4.3 again) that trusted names may be sent along an untrusted channel, as suggested in the intuitions at the beginning of Section 3.3. Note however that due to the formation of the input rule, and the reverse direction of the sub-typing clause in that rule, any such names on an untrusted channel will be bound to untrusted variables at their destination.

Note especially that there is no sub-typing clause present in the certify rule (in fact, it is unchanged from the rule in System $\mathcal{T}_{\mathsf{FO}\pi}$). This is mostly as

a result of the definition of sub-typing only involving the outer-most annotations and not a structural component; since the annotation on the bound variable is determined by the form of the rule (requiring a trusted variable in one branch and an untrusted variable in the other), a sub-typing clause does not make sense.

3.5 First-Order Security System With Sub-Typing

The previous two systems both can provide integrity of data against being over-written with lower-integrity values, even in the presence of run-time coercion. They are still susceptible though to an implicit attack, where an untrusted value can affect the execution of a trusted computation, thus the security they provide is not total. For example, consider the following program:

$$x.y.\mathbf{0}$$

Assume that x is untrusted and y is trusted. In this case then, the use of the channel y is dependent on x first being used, and thus a trusted channel relies on a low integrity value for its correct usage. The untrusted channel may be used at the wrong time, or take a longer time to execute, thus disrupting synchronous processes, and so on.

Thus, for a system to be considered secure against low integrity data it must be shown that untrusted values do not affect the successful execution of trusted computations. The final step in this presentation of first-order systems is to include a security level in the deduction, which specifies the trustedness of channels that may be used. It can then be demonstrated (see Chapter 4) that values of a lower trustedness than the security level are unable to disrupt execution.

The rules implementing this system, collectively referred to as System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$, are presented in Figure 3.5. They make use of a different judgement form again, described in Definition 3.5.1:

Definition 3.5.1 (Judgements in $\mathcal{T}_{\mathsf{FO}'\pi\leq}$) *Judgements in $\mathcal{T}_{\mathsf{FO}'\pi\leq}$ take the following two forms, with the usual meaning of well-formed process and well-formed null, sum, or guarded process respectively:*

$$\Gamma \ \vdash^{b}_{\mathrm{FO}'\leq} \ \mathrm{P} : \mathsf{Proc}$$
$$\Gamma \ \vdash^{b}_{\mathrm{FO}'\leq} \ \mathrm{P} : \mathsf{Proc}$$

There is now an additional annotation, attached to the turn-style. This represents the trustedness at which the process is typed under, and plays the

same role as the "program counter" level (traditionally so-called because it represents the security level reached by executing the program) in many information flow systems. It represents the least trustedness level of channels that may be used by that process, and will be referred to as the security level or the security level judgement annotation.

$$\vdash_{\mathrm{FO}'\leq} \sigma^b$$

has the usual meaning of a well-formed type under System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$.

The following Lemma states that every judgement in System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$ also forms a valid judgement in System $\mathcal{T}_{\mathsf{FO}\pi\leq}$ by ignoring the security level:

Lemma 3.5.2 $\forall b, \Gamma, \mathrm{P}.$

$$\Gamma \vdash^b_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc} \Longrightarrow \Gamma \vdash_{\mathrm{FO}\leq} \mathrm{P} : \mathsf{Proc}$$

Proof. By induction on the respective derivations, noting that the rules in each case are identical, but with stricter requirements in System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$. □

Note that a similar result can be stated in the opposite direction, for example for a deduction $\Gamma \vdash_{\mathrm{FO}\leq} \mathrm{P} : \mathsf{Proc}$ in System $\mathcal{T}_{\mathsf{FO}\pi\leq}$ there is some security level b such that $\Gamma \vdash^b_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}$ is a valid deduction in System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$. In the extreme case this holds by letting $b = \mathrm{U}$, so this result is much less useful and interesting.

No changes are required for the typed labelled transition semantics.

3.5.1 Analysis of the Rules

The general form, as could be expected, remains more or less unchanged from the rules of System $\mathcal{T}_{\mathsf{FO}\pi\leq}$. Two categories of change are worth pointing out:

- what happens when processes are combined, and
- the new restriction on channels that may be used.

Combination of separate processes occurs in two rules (certify will be discussed separately); those for composition and summation. Both require that the resultant security level is the infinum or meet of the two antecedent levels. This seems reasonable for composition where the processes are potentially independent, so the conservative choice is taken for the combination. A more logical-sounding choice for summation might be to require the security levels to match, since the summation will act like exactly one of its summand processes. However, the presence of a rule to lower the security level (see below)

$$\frac{\vdash_{\mathrm{FO}'\leq} \overrightarrow{\sigma^b}}{\vdash_{\mathrm{FO}'\leq} (c \cdot \overrightarrow{\sigma^b})^c}\ (type.) \qquad \frac{\Gamma \vdash^b_{(\mathrm{FO}'\leq)} \mathrm{P} : \mathsf{Proc}}{\Gamma \vdash^b_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}}\ (conv.)$$

$$\frac{}{\emptyset \vdash^b_{(\mathrm{FO}'\leq)} \mathbf{0} : \mathsf{Proc}}\ (zero) \qquad \frac{\Gamma \vdash^b_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc} \quad c \leq_{\mathsf{B}} b}{\Gamma \vdash^c_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}}\ (rel.)$$

$$\frac{\Gamma \vdash^b_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}}{\Gamma \vdash^b_{\mathrm{FO}'\leq} !\mathrm{P} : \mathsf{Proc}}\ (rep.) \qquad \frac{\Gamma_x, x : \sigma^b \vdash^c_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}}{\Gamma_x \vdash^c_{\mathrm{FO}'\leq} (\nu x : \sigma^b)\mathrm{P} : \mathsf{Proc}}\ (res.)$$

$$\frac{\Gamma \vdash^b_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc} \quad x \notin \mathrm{dom}\Gamma \quad \vdash_{\mathrm{FO}'\leq} \sigma^c}{\Gamma, x : \sigma^c \vdash^b_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}}\ (weak.)$$

$$\frac{\Gamma \vdash^b_{(\mathrm{FO}'\leq)} \mathrm{P} : \mathsf{Proc} \quad \Theta \vdash^c_{(\mathrm{FO}'\leq)} \mathrm{Q} : \mathsf{Proc}}{\Gamma, \Theta \vdash^{bc}_{(\mathrm{FO}'\leq)} \mathrm{P} + \mathrm{Q} : \mathsf{Proc}}\ (sum.)$$

$$\frac{\Gamma \vdash^b_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc} \quad \Theta \vdash^c_{\mathrm{FO}'\leq} \mathrm{Q} : \mathsf{Proc}}{\Gamma, \Theta \vdash^{bc}_{\mathrm{FO}'\leq} \mathrm{P}|\mathrm{Q} : \mathsf{Proc}}\ (comp.)$$

$$\frac{\Gamma_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma'^{b'}} \vdash^d_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc} \quad \overrightarrow{\sigma^b} \leq \overrightarrow{\sigma'^{b'}} \quad d \leq_{\mathsf{B}} c}{\Gamma_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^b})^c \vdash^d_{(\mathrm{FO}'\leq)} x(\overrightarrow{y} : \overrightarrow{\sigma^b}).\mathrm{P} : \mathsf{Proc}}\ (inp.)$$

$$\frac{\Gamma, x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma'^{b'}} \vdash^d_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc} \quad \overrightarrow{\sigma'^{b'}} \leq \overrightarrow{\sigma^b} \quad d \leq_{\mathsf{B}} c}{\Gamma, x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma'^{b'}} \vdash^d_{(\mathrm{FO}'\leq)} \overline{x}[\overrightarrow{y}].\mathrm{P} : \mathsf{Proc}}\ (out.)$$

$$\frac{\begin{array}{ll}\Gamma_y, x : (\sigma^i)^b, y : \sigma^{\mathrm{T}} \vdash^c_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc} & c \leq_{\mathsf{B}} b \\ \Theta_y, x : (\sigma^i)^b, y : \sigma^{\mathrm{U}} \vdash^d_{\mathrm{FO}'\leq} \mathrm{Q} : \mathsf{Proc} & d \leq_{\mathsf{B}} b\end{array}}{\Gamma_y, \Theta_y, x : (\sigma^i)^b \vdash^{ic+id}_{(\mathrm{FO}'\leq)} x(y : \sigma^i)_{\underset{\mathrm{certify}}{?}}\mathrm{P} \oplus \mathrm{Q} : \mathsf{Proc}}\ (cert.)$$

Figure 3.5: Type Rules Of System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$

renders this a moot point, as requiring the levels to match then merely has the effect of adding an extra step to a derivation. Thus, the more pragmatic meet strategy is chosen instead.

There is a new rule (*rel.*) to allow the security level to be relaxed. Because the other restrictions require the name of a guard to be at least as trusted as the security level, the transitivity of $\leq_{\mathsf{B}}$ means this stipulation is not affected by the relaxation. Note that even in the absence of such a rule its effect could still be achieved, with the composition rule weakening the combined security level, to form a behaviourally identical process with a lower security level. For example P|**0** behaves identically to P, and will have a security level which is the lower of the two.

The main security property is a result of the form of the rules governing construction of guarded processes, in other words input, output, and certify. All require that the channel has an integrity level at least as trusted as the security level of the process they prefix.

The rule for certify is the most interesting. It has the same requirements that the input channel must be at least as trusted as the process annotations, but the difference lies in the combination of the two process annotations. Recall that both a summation and a composition required that the combined security level was the meet of those of the antecedents, with the reasoning being that this was a conservative approach for the composition where the processes were independent, and allowed more processes to be typed in the case of the non-deterministic choice present in a summation. Neither situation is represented in a certify construct: there is a choice involved, but not a non-deterministic one. Since the reduction implies a coercion and the direction of the coercion indicates which branch is selected, there is enough information to create a combined annotation that accurately reflects both levels in the subsequent reduction. This is achieved by using the variable annotation — which will be coerced to a concrete value during the reduction — to select the corresponding security level:

$$i \cdot c + \bar{i} \cdot d$$

Observe that in the event that i is determined to be trusted by the certify oracle this expression will reduce to c, the security level of the process that will concurrently be reduced, and conversely to d if it is found to be untrusted.

3.6 Higher-Order System With Sub-Typing

The previous systems (of Sections 3.3, 3.5, and 3.4) concerned the same core language, with progressive extensions to the type system to enable more

programs to be typed. All systems though — since they were based on the same language — by definition were first-order; they are only useful for analysing systems involving simple data transmission. This is obviously still extremely useful, and still very powerful since the networks may dynamically alter their topologies, but the fact remains that many contemporary systems are being constructed around the premise of mobile *code* as well as data. This is an important field (for reasons outlined in Section 2.2.3), and so this section introduces a type-based tool for analysing such systems.

New questions not present in the first-order systems immediately arise, and the type rules must naturally be extended to answer them. Intuitively, since processes may be transmitted — including along untrusted channels — they also run a risk of being compromised and thus it seems natural to associate a trustedness value with processes as well as names. This also suggests that perhaps code and data arising *from* such an untrusted process must also be untrusted, even over a trusted channel of communication (if a process is for some reason regarded as malicious, it seems natural that all messages from it should be distrusted).

A far more subtle issue arises however when considering what exactly it means for a process to be "untrusted". For example, should a malicious program, running in isolation from the rest of the system (for example, running inside a complete sandbox and unable to interact with other programs or the file system), still be considered "untrusted"? The view held in this thesis is that it should not; that the final trustedness of a program *must* be deduced with consideration given to the communication context in which it is executed.

The situation is complicated further still by the realisation that processes which are "trusted" in isolation may none-the-less be used by malicious processes to communicate with the host environment, and thus *in that setting* should be untrusted.

The following Example (3.6.1) illustrates this point:

Example 3.6.1 *Assume process* P, Q, *and* R *that are closed with respect to each other; that is* $FN(\mathrm{P}) \cap FN(\mathrm{Q}) \cap FN(\mathrm{R}) = \emptyset$. *Create from* P *a trusted agent* $\mathrm{P}' \triangleq x(v).\overline{z}[v].\mathrm{P}$. *Similarly, assume the following untrusted agent:* $\mathrm{Q}' \triangleq \overline{y}[w].\mathrm{Q}$. *With the port of observation (the only channel through which the external environment is accessible) limited to* x, *it is easy to informally verify that no possibility for corruption exists: the two processes are isolated from each other, sharing no channels of communication, and only the trusted process communicates with the host system.*

If a third agent is introduced into the system the situation may change, for example as follows: given $\mathrm{R}' \triangleq z(v).y(w).\mathrm{R}$ *then no matter what the*

trustedness of the new process, it has introduced an avenue of communication (between P' *and* R' *using* z*, and between* R' *and* Q' *via* y*) between the (initially trusted) first process* P' *(and hence the outside environment) and the malicious process, so the entire group should be conservatively classified as untrusted. In other words, from the perspective of the host untrustedness is transitive across processes sharing names.*

Implementing this perspective is not trivial, and the solution, by necessity, introduces some novel concepts. There are two additional components to the new system:

- Each deduction in the system is performed in relation to the *external environment*: informally, this is a set of names that the process under scrutiny may use to communicate with the host system if the process shares any of these names (that is, if the process *does not* have any knowledge of any of these names then it must eventually be considered trusted as it is isolated from the external system). It corresponds to the security property of the host running the program.

- Additional information is maintained about the *context* each name appears in. Informally, this is related to the trustedness of the process using the name; for example, an untrusted process may still communicate (and potentially with the environment) using a trusted channel, but that channel is used in an *untrusted context*. In other words, while the channel in question preserves the integrity of the data it carries it may still be used by a malicious program to communicate with the external system and perhaps inject false data, and thus is considered to be in an untrusted context. Perhaps a second way of looking at it is whilst the type annotations track *explicit* trustedness (that is, whether a channel corrupts its contents or not), contexts track *implicit* trust: essentially, if channels are being used for covert purposes or not.

3.6.1 Preliminaries

Several new concepts need to be introduced before the new system may be presented; these are considered in this section.

First of all, there is a small change to the syntax, to allow the programmer to declare the integrity of processes. This is implemented by attaching an annotation to the null process. See Definition 3.6.2 for the syntax in full.

Definition 3.6.2 (Annotated Higher-Order Syntax)

$$\begin{array}{rcl} \mathrm{P} & ::= & \mathbf{0}^b \mid \sum_{i\in I} \pi_i.\mathrm{P}_i \mid !\mathrm{P} \mid \mathrm{P}|\mathrm{P} \mid (\nu x : \sigma^b)\mathrm{P} \mid X \mid X\langle\overrightarrow{K}\rangle \mid \mathrm{F}\langle\overrightarrow{K}\rangle \\ & & \mid x(V : \sigma^i)_{\mathrm{certify}}^{?}\mathrm{P} \oplus \mathrm{P} \\ \mathrm{F} & ::= & (\overrightarrow{V} : \overrightarrow{\sigma^b})\mathrm{P} \\ \mathrm{A} & ::= & \mathrm{P} \mid \mathrm{F} \\ \pi & ::= & x(\overrightarrow{V} : \overrightarrow{\sigma^b}) \mid \overline{x}[\overrightarrow{V}] \end{array}$$

Definition 3.6.3 (Higher-Order Execution Contexts) *Write $\mathbb{C}$ to represent the context created as part of a deduction. It is a partial function from names/variables to annotations (trust expressions; that is $\mathbb{C} \subseteq \mathcal{V} \times \mathcal{B}_{\mathcal{A}}$); the domain of a context is exactly the free variables of the process in the deduction to which it pertains.*

1. *As usual, $\mathbb{C}_V$ represents the context $\mathbb{C}$ with the name or variable V removed from its domain.*

2. *The construct $\mathbb{C}_{\mathrm{T}}$ is the total function derived from $\mathbb{C}$ as:*

$$\mathbb{C}_{\mathrm{T}}(V) \;=\; \begin{cases} \mathbb{C}(V) & , \; V \in \mathrm{dom}\mathbb{C} \\ \mathrm{T} & , \; \text{otherwise} \end{cases}$$

3. *A context is* compatible *with another, written $\mathbb{C}_1 \asymp \mathbb{C}_2$, if and only if for each name or variable V in the domain of both $\mathbb{C}_1$ and $\mathbb{C}_2$ then $\mathbb{C}_1(V) = \mathbb{C}_2(V)$.*

4. *Contexts may be combined using the ',' operator; this creates the union and is defined if and only if the two are compatible:*

$$(\mathbb{C}_1, \mathbb{C}_2)(x) \;=\; \begin{cases} \mathbb{C}_1(x) & , \; x \in \mathrm{dom}\mathbb{C}_1 \\ \mathbb{C}_2(x) & , \; x \in \mathrm{dom}\mathbb{C}_2 \end{cases}$$

   The ',' operator is symmetric and transitive. As with environments (see Section 2.3.1) union with a singleton context will frequently be abbreviated to $\mathbb{C}, x : b$. Given a sequence $\overrightarrow{V} = V_1 \ldots V_n$, define $\overrightarrow{V} : b$ as $V_1 : b \ldots V_n : b$

5. *The operator $\subseteq$ is the precise subset operator:*

$$\mathbb{C}_1 \subseteq \mathbb{C}_2 \iff \forall x \in \mathrm{dom}\mathbb{C}_1.\mathbb{C}_1(x) = \mathbb{C}_2(x)$$

6. *Multiplication on contexts is defined as follows:*

$$b \cdot \mathbb{C} \triangleq \{V : b \cdot \mathbb{C}(V) \mid V \in \mathrm{dom}\mathbb{C}\}$$

This is extended to sequences of contexts in the logical way; that is $b \cdot \overrightarrow{\mathbb{C}} = b \cdot \mathbb{C}_1 \ldots b \cdot \mathbb{C}_n$ *if* $\overrightarrow{\mathbb{C}} = \mathbb{C}_1 \ldots \mathbb{C}_n$.

7. *Given an annotation b, a ternary operator on a pair of contexts may be defined as follows (the motivation for this will become apparent later on):*

$$(\mathbb{C}_1 \langle b \rangle \mathbb{C}_2)(V) = \begin{cases} b \cdot \mathbb{C}_1(V) + \overline{b} \cdot \mathbb{C}_2(V) & , \text{ if } V \in \mathrm{dom}\mathbb{C}_1 \cap \mathrm{dom}\mathbb{C}_2 \\ b \cdot \mathbb{C}_1(V) + \overline{b} & , \text{ if } V \in \mathrm{dom}\mathbb{C}_1 \wedge V \notin \mathrm{dom}\mathbb{C}_2 \\ b + \overline{b} \cdot \mathbb{C}_2(V) & , \text{ if } V \in \mathrm{dom}\mathbb{C}_2 \wedge V \notin \mathrm{dom}\mathbb{C}_1 \end{cases}$$

The $\langle\rangle$ *operator is* not *symmetric. It binds tighter than ',' so* $\mathbb{C}_1, \mathbb{C}_2 \langle b \rangle \mathbb{C}_3$ *is equivalent to* $\mathbb{C}_1, (\mathbb{C}_2 \langle b \rangle \mathbb{C}_3)$.

Definition 3.6.4 (External Context) *Write* $\mathbb{C}_\epsilon$ *to represent the* interface *to the host system that any process is typed under. It is a finite — possibly empty — set of channels (that is,* $\mathbb{C}_\epsilon \subseteq \mathcal{N}$*). A totally empty external context corresponds to the situation of a total sandbox; the process under investigation has* no *avenue of communication with the host system and is thus completely isolated.* $\mathbb{C}_\epsilon$ *is constant throughout a deduction tree.*

Definition 3.6.5 (Annotated Type Syntax) *The type syntax must also be extended in a higher-order system, due to the need to differentiate between instantiation to a name and instantiation to a process (Vasconcelos 1993). This is achieved by taking the symbol* Proc*, formerly just used to denote a well-formed judgement, and admitting it as a type for well-formed processes. Because processes now have an integrity level associated, an annotation is also attached. Abstractions are represented in a manner reminiscent of function types using an arrow, preceded by a possibly-empty sequence of the types of the names and variables abstracted on, enclosed in parentheses. As before,* σ *ranges over base types.*

The new type syntax is:

$$\sigma ::= (\overrightarrow{\sigma})^b \mid \mathsf{Proc}^b \mid (\overrightarrow{\sigma}) \to \mathsf{Proc}^b$$

Multiplication and related definitions are extended logically to handle process types as well. So, $b \cdot ((\overrightarrow{\sigma^c}) \to \mathsf{Proc}^d)$ *is interpreted as* $((\overrightarrow{\sigma^c}) \to \mathsf{Proc}^{bd}$. *The*

same convention of usually writing the type with the outmost annotation displayed is also adopted here; note that the annotation displayed refers to the last part of the term, for example the type $(()^c) \rightarrow \mathsf{Proc}^b$ *might be referred to in the discussion as* σ^b *(not* σ^c *for example).*

Definition 3.6.6 (Sub-Typing) *The definition of sub-typing is also extended, although in a limited way. For reasons explained later, sub-typing on processes is not admitted into the system, thus the relation below only includes a reflexive relation on process types. Sub-Typing on channel types remains the same (including the restriction that it only considers the outermost annotation due to a lack of usage information; see Definition 3.4.3 on page 58 for an overview).*

$$\frac{c \leq_{\mathsf{B}} b}{(\overrightarrow{\sigma^d})^b \leq (\overrightarrow{\sigma^d})^c} \quad \frac{}{\mathsf{Proc}^b \leq \mathsf{Proc}^b} \quad \frac{}{(\overrightarrow{\sigma^b}) \rightarrow \mathsf{Proc}^c \leq (\overrightarrow{\sigma^b}) \rightarrow \mathsf{Proc}^c}$$

Definition 3.6.7 (Higher-Order Judgements) *A type judgement in System* $\mathcal{T}_{\mathsf{HO}\pi\leq}$ *usually has one of the forms*

$$\Gamma \quad \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \quad \mathrm{K} : \sigma^b; \mathbb{C}$$
$$\Gamma \quad \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \quad \mathrm{P} : \mathsf{Proc}^b; \mathbb{C}$$

As before, Γ *is the typing environment,* K *the agent being typed, and* σ^b *the resultant type of the (well-formed) agent under the assumptions* Γ*. For reasons of convenience that will be elaborated later, the definition is relaxed to allow names to appear on the right-hand side. This is purely for convenience, and the majority of the time this form of judgement will refer to an agent. The second form is only used for guarded, null, and summation processes as usual. The new components are*

- $\vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq}$*, the new 'turn-style' operator which also states under what external environment (that is,* $\mathbb{C}_\epsilon$*) the agent is being typed; and*
- $\mathbb{C}$*, the context of each (free) name in the deduction (see Definition 3.6.3). Note that it is a product of that particular deduction, rather than a set of assumptions, so is written on the right-hand side of the deduction.*

An additional form of judgement is

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b$$

which is identical to the first form, but missing the context information. Only one rule in system $\mathcal{T}_{\mathsf{HO}\pi\leq}$ *has a judgement of this form in the consequent (and*

it never appears as an antecedent); it represents the very final deduction of a given process in a given environment, with the contextual information utilised (to calculate the overall trustedness under $\mathbb{C}_\epsilon$) and hence discarded.

A final form

$$\vdash_{\mathrm{HO}\leq} \sigma^b$$

states that the type σ^b is well-formed under the rules of System $\mathcal{T}_{\mathsf{HO}\pi\leq}$.

As before, the vector short-hand will be exploited where useful, such as

$$\overrightarrow{\Gamma} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma^b}; \overrightarrow{\mathbb{C}}$$

to refer to the possibly-empty sequence of judgements

$$\Gamma_1 \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} K_1 : \sigma_1^{b_1}; \mathbb{C}_1 \ldots \Gamma_n \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} K_n : \sigma_n^{b_n}; \mathbb{C}_n$$

In addition, subscripts will occasionally be used to refer to range over judgements in a sequence, for example $\forall \Gamma_i, \mathbb{C}_i \ldots$ meaning that the statement to follow should apply to each Γ_i, $\mathbb{C}_i$ pair from the sequence of judgements.

3.6.2 Structure Of The Rules

Definition 3.6.8 *Define the notational convenience* $\mathsf{chans}\left(\overrightarrow{K}\right)$ *as the* names *(or channels, as the notation represents) of the sequence $\overrightarrow{K}$:*

$$\begin{aligned} \mathsf{chans}(\cdot) &= \emptyset \\ \mathsf{chans}\left(x\overrightarrow{K}\right) &= \{x\} \cup \mathsf{chans}\left(\overrightarrow{K}\right) \\ \mathsf{chans}\left(\mathrm{A}\overrightarrow{K}\right) &= \mathsf{chans}\left(\overrightarrow{K}\right) \end{aligned}$$

For example, $\mathsf{chans}(x\mathrm{P}y\mathrm{F}) = \{x, y\}$.

The higher-order sub-typing rules are referred to collectively as system $\mathcal{T}_{\mathsf{HO}\pi\leq}$ and are presented in Figures 3.7 and 3.8. For ease of introduction, the simpler rules (mostly the structural rules) are presented first, with the notationally-complicated cases following.

While much similarity with the first-order systems remains there is also considerable divergence so some more detailed explanation is required.
Firstly the zero rule; the main point worthy of note is the empty context: recall that the context of a deduction contains exactly the free names of the agent in the deduction, so this is quite natural. For the purposes of a deduction, an inactive process may be given any trustedness level. The weaken rule is also relatively familiar; observe that the context is unchanged

$$\frac{}{\vdash_{\mathrm{HO}\leq} \mathsf{Proc}^b}\ (type-proc) \qquad \frac{\vdash_{\mathrm{HO}\leq} \overrightarrow{\sigma^b}}{\vdash_{\mathrm{HO}\leq} (c\cdot\overrightarrow{\sigma^b})^c}\ (type-chan)$$

$$\frac{\vdash_{\mathrm{HO}\leq} \overrightarrow{\sigma^b}}{\vdash_{\mathrm{HO}\leq} (\overrightarrow{\sigma^b})\to \mathsf{Proc}^b}\ (type-abs)$$

Figure 3.6: Type Formation Rules for System $\mathcal{T}_{\mathsf{HO}\pi\leq}$

$$\frac{}{\emptyset \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \mathbf{0}^b : \mathsf{Proc}^b; \emptyset}\ (zero.) \qquad \frac{\Gamma \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \mathrm{P} : \mathsf{Proc}; \mathbb{C}}{\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}; \mathbb{C}}\ (conv.)$$

$$\frac{\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{A} : \sigma^b; \mathbb{C} \quad \vdash_{\mathrm{HO}\leq} \sigma'^c \quad V \notin \mathrm{dom}\Gamma}{\Gamma, V : \sigma'^c \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{A} : \sigma^b; \mathbb{C}}\ (weak.)$$

$$\frac{\vdash_{\mathrm{HO}\leq} \sigma^b}{x : \sigma^b \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} x : \sigma^b; \emptyset}\ (var-1) \qquad \frac{\vdash_{\mathrm{HO}\leq} \sigma^b}{X : \sigma^b \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} X : \sigma^b; X : b}\ (var-2)$$

$$\frac{\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C}}{\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} !\mathrm{P} : \mathsf{Proc}^b; \mathbb{C}}\ (repl.)$$

$$\frac{\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_1 \quad \Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{Q} : \mathsf{Proc}^c; \mathbb{C}_2}{\Gamma, \Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P}|\mathrm{Q} : \mathsf{Proc}^{bc}; \mathbb{C}_1, \mathbb{C}_2}\ (comp.)$$

$$\frac{\Gamma_x, x : \sigma^c \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C} \quad x \notin \mathbb{C}_\epsilon}{\Gamma_x \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} (\nu x : \sigma^c)\mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_x}\ (res.)$$

$$\frac{\Gamma \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_1 \quad \Theta \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \mathrm{Q} : \mathsf{Proc}^c; \mathbb{C}_2}{\Gamma, \Theta \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \mathrm{P} + \mathrm{Q} : \mathsf{Proc}^{bc}; \mathbb{C}_1, \mathbb{C}_2}\ (sum.)$$

Figure 3.7: Type Rules Of System $\mathcal{T}_{\mathsf{HO}\pi\leq}$

$$\frac{\Gamma_{\overrightarrow{V}}, \overrightarrow{V} : \overrightarrow{\sigma^c} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C} \quad \forall V \in \overrightarrow{V}.\mathbb{C}_{\mathrm{T}}(V) \geq_{\mathsf{B}} b, V \notin \mathbb{C}_\epsilon}{\Gamma_{\overrightarrow{V}} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} (\overrightarrow{V} : \overrightarrow{\sigma^c})\mathrm{P} : (\overrightarrow{\sigma^c}) \to \mathsf{Proc}^b; \mathbb{C}_{\overrightarrow{V}}} \ (abs.)$$

$$\frac{\begin{array}{c}\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{A} : (\overrightarrow{\sigma^c}) \to \mathsf{Proc}^b; \mathbb{C}_1 \quad \overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma^d}; \overrightarrow{\mathbb{C}_2} \quad \overrightarrow{\sigma^d} \leq \overrightarrow{\sigma^c} \\ \forall \mathbb{C}_{2i}, V \in \mathrm{dom}\mathbb{C}_{2i}.\mathbb{C}_{2i}(V) \leq_{\mathsf{B}} d_i \quad \forall x \in \mathbb{C}_\epsilon.\overrightarrow{\mathbb{C}_{2\mathrm{T}}}(x) \geq_{\mathsf{B}} b\end{array}}{\Gamma, \overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{F}\langle\overrightarrow{K}\rangle : \mathsf{Proc}^b; \mathbb{C}_1, \overrightarrow{\mathbb{C}_2}, \mathsf{chans}\left(\overrightarrow{K}\right) : b} \ (app.)$$

$$\frac{\begin{array}{c}\Gamma_{\overrightarrow{V}}, x : (\overrightarrow{\sigma^c})^b, \overrightarrow{V} : \overrightarrow{\sigma^{c'}} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^d; \mathbb{C} \\ \overrightarrow{\sigma^c} \leq \overrightarrow{\sigma^{c'}} \quad \forall V \in \overrightarrow{V}.\mathbb{C}_{\mathrm{T}}(V) \geq_{\mathsf{B}} d, V \notin \mathbb{C}_\epsilon\end{array}}{\Gamma_{\overrightarrow{V}}, x : (\overrightarrow{\sigma^c})^b \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} x(\overrightarrow{V} : \overrightarrow{\sigma^{c'}}).\mathrm{P} : \mathsf{Proc}^d; \mathbb{C}_{\overrightarrow{V}}, x : d} \ (inp.)$$

$$\frac{\begin{array}{c}\Gamma, x : (c \cdot \overrightarrow{\sigma^d})^b \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^c; \mathbb{C}_1 \quad \overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma^{d'}}; \overrightarrow{\mathbb{C}_2} \\ \overrightarrow{\sigma^{d'}} \leq c \cdot \overrightarrow{\sigma^d} \quad \forall x \in \mathbb{C}_\epsilon.\overrightarrow{\mathbb{C}_{2\mathrm{T}}}(x) \geq_{\mathsf{B}} c \quad \forall \mathbb{C}_{2i}, V \in \mathrm{dom}\mathbb{C}_{2i}.\mathbb{C}_{2i}(V) \leq_{\mathsf{B}} d'_i\end{array}}{\Gamma, x : (c \cdot \overrightarrow{\sigma^d})^b, \overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \overline{x}[\overrightarrow{K}].\mathrm{P} : \mathsf{Proc}^c; \mathbb{C}_1, x : c, \overrightarrow{\mathbb{C}_2}, \mathsf{chans}\left(\overrightarrow{K}\right) : c} \ (out.)$$

$$\frac{\begin{array}{c}\Gamma_V, V : \sigma^{\mathrm{T}}, x : (\sigma^i)^e \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^c; \mathbb{C}_1 \quad \mathbb{C}_{1\mathrm{T}}(V) \geq_{\mathsf{B}} c \\ \Theta_V, V : \sigma^{\mathrm{U}}, x : (\sigma^i)^e \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{Q} : \mathsf{Proc}^d; \mathbb{C}_2 \quad \mathbb{C}_{2\mathrm{T}}(V) \geq_{\mathsf{B}} d \quad V \notin \mathbb{C}_\epsilon\end{array}}{\begin{array}{l}\Gamma_V, \Theta_V, x : (\sigma^i)^e \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} x(V : \sigma^i)_{\underset{\text{certify}}{?}}\mathrm{P} \oplus \mathrm{Q} : \mathsf{Proc}^{i\cdot c + \bar{i}\cdot d}; \\ \qquad\qquad \mathbb{C}_{1V} \langle i \rangle \mathbb{C}_{2V}, x : (i \cdot c + \bar{i} \cdot d)\end{array}} \ (cert.)$$

$$\frac{\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C}}{\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{\bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}_{\mathrm{T}}(x)}} \ (final)$$

Figure 3.8: Type Rules of System $\mathcal{T}_{\mathsf{HO}\pi\leq}$(continued)

in the consequent though, since it is only defined by the free variables of the agent in the deduction, and not the environment.

Notice that there are now *two* variable introduction rules (var-1 and var-2); one for names and one for variables. No such rules were present in the first-order language, so it is worth considering the reasons for their existence in a higher-order setting. The introduction rule for a variable (var-2) is necessary because of a process such as $x(X).X$, in which case a rule to introduce X on the right-hand side is required (formerly, all deductions would have used at least one instance of the zero rule; now, processes may also be built up on top of variables). Its image in the context is its own annotation, as it is a free variable in a process (itself) with this trustedness. The inclusion of the rule var-1 is driven mainly by convenience, as will become apparent when the output rule is discussed (note though that it is *not* an agent and doesn't appear in any context, so the context remains empty).

The rules for replication and restriction are fairly innocuous; the only point worthy of note is that since restriction (which may also be on a variable as well as a name) is a binding operation the bound name/variable is removed from both the assumption set and the associated context (which must contain only the free variables) — a little thought reveals that this is logical since it creates a *local* variable and is thus unable to communicate with the external environment.

The first rule to be discussed in-depth is the rule Final:

$$\frac{\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C}}{\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{\bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}_\mathrm{T}(x)}}$$

As its name suggests, it is always the final rule in any complete deduction: it combines the information collected in the contexts during the deduction, and uses it to determine the possibility of a malicious agent communicating with the host system.

The motivation for this is briefly recapped: consider two processes, one trusted and the other untrusted. They are isolated from each other, and further only the trusted process can communicate with the host so this would seem to be a safe situation. The composition rule, discussed shortly, would type this as untrusted however, by taking the conservative approach of the greatest-lower-bound on the combined trustedness. The rule Final rectifies this situation by computing the overall trustedness with respect to the host. This is what the expression

$$\bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}_\mathrm{T}(x)$$

does; it computes the greatest lower bound of the context each name able to communicate with the host (that is, all those in $\mathbb{C}_\epsilon$) appears under.[2]

Some trivial examples illustrate these points and the use of the final rule.

Example 3.6.9 *Assume the following deduction:* $\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{\mathrm{U}}; \emptyset$. *That is, an untrusted process, that is* closed *or has no free variables (note the empty context). Then by definition* $\bigwedge_{x \in \mathbb{C}_\epsilon} \emptyset_{\mathrm{T}}(x) = \mathrm{T}$ *for all* $\mathbb{C}_\epsilon$*, and thus via the final rule* $\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{\mathrm{T}}$*; that is, the final deduction states that the process is trusted — a logical result, since it cannot interact with anyone (especially the host) and is thus harmless.*

Example 3.6.10 *Now consider the following deduction of an untrusted process:* $\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{\mathrm{U}}; \{y : \mathrm{U}\}$. *Assume that the deduction has been performed with respect to a host that leaves the single channel* x *open; that is* $\mathbb{C}_\epsilon = \{x\}$. *Note that the process in this example is not closed, and exposes just a single name* y *(in an untrusted context). Thus it still has no means of communication with the host, and can therefore be trusted; the final rule confirms this:* $\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{\mathrm{T}}$*, since if* $\mathbb{C}_\epsilon = \{x\}$ *then* $\bigwedge_{x \in \mathbb{C}_\epsilon} \{y : \mathrm{U}\}_{\mathrm{T}}(x) = \mathrm{T}$*, as expected.*

Alternatively, if the deduction had been performed under $\mathbb{C}_\epsilon = \{y\}$ *then* $\bigwedge_{x \in \mathbb{C}_\epsilon} \{y : \mathrm{U}\}_{\mathrm{T}}(x) = \mathrm{U}$ *and thus* $\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{\mathrm{U}}$.

Example 3.6.11 *Example 3.6.1 informally described the following program of three processes:*

$$x(v).\overline{z}[v].\mathrm{P} \mid \overline{y}[w].\mathrm{Q} \mid z(v).y(w).\mathrm{R}$$

It can now be shown (more formally than before) how typing would proceed for the same program, and thus why it would be rejected as it currently stands. Assume the following three deductions have already been made:

$$\begin{aligned} \Gamma &\vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{\mathrm{T}}; \mathbb{C}_P \\ \Gamma' &\vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{Q} : \mathsf{Proc}^{\mathrm{U}}; \mathbb{C}_Q \\ \Gamma'' &\vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{R} : \mathsf{Proc}^{\mathrm{T}}; \mathbb{C}_R \end{aligned}$$

Because each process is closed with respect to the others it must be the case that $\mathbb{C}_P \cap \mathbb{C}_Q \cap \mathbb{C}_R = \emptyset$*, recalling that a context contains exactly the* free *names*

[2] There is the possibility that $\mathbb{C}_\epsilon$ contains *no* names; in other words, a perfect sandbox. In this case the expression evaluates to T, following systems such as Haskell that define conjunction over an empty list as true.

and variables of a process. The complete deductions for each branch may now be formed, using applications of the input and output rules as appropriate:

$$\begin{aligned}\Gamma_v &\vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} x(v).\overline{z}[v].\mathrm{P} : \mathsf{Proc}^{\mathrm{T}}; \mathbb{C}_P, x:\mathrm{T}, z:\mathrm{T}\\ \Gamma' &\vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overline{y}[w].\mathrm{Q} : \mathsf{Proc}^{\mathrm{U}}; \mathbb{C}_Q, w:\mathrm{U}, y:\mathrm{U}\\ \Gamma''_{vw} &\vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} z(v).y(w).\mathrm{R} : \mathsf{Proc}^{\mathrm{T}}; \mathbb{C}_R, y:\mathrm{T}, z:\mathrm{T}\end{aligned}$$

Now, compose (using the composition rule, and assuming $\mathbb{C}_{PR} = \mathbb{C}_P, \mathbb{C}_R$) the P *and* R *branches:*

$$\Gamma_v, \Gamma''_{vw} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} x(v).\overline{z}[v].\mathrm{P} \mid z(v).y(w).\mathrm{R} : \mathsf{Proc}^{\mathrm{T}}; \mathbb{C}_{PR}, x:\mathrm{T}, y:\mathrm{T}, z:\mathrm{T}$$

So far so good; now — again using the composition rule — combine with the Q *branch: here however lies a problem, as this would involve the union of the contexts $\mathbb{C}_{PR} = \{x:\mathrm{T}, y:\mathrm{T}, z:\mathrm{T}\}$ and $\mathbb{C}_Q = \{w:\mathrm{U}, y:\mathrm{U}\}$ which is undefined as the image of y is* T *in one branch and* U *in the other.*

Thus this program may only be typed if all processes are typed as untrusted, as communication is transitive. (It may be checked that the order of composition is immaterial).

Example 3.6.12 *Consider Example 3.6.11 again, to see why context union is defined as it is. An alternative definition, which at first glance appears more flexible, might be to* multiply *the images of each name appearing in both contexts: so if $\mathbb{C}_1 = \{x:\mathrm{T}, y:\mathrm{T}\}$ and $\mathbb{C}_2 = \{y:\mathrm{U}, z:\mathrm{U}\}$ then $\mathbb{C}_1, \mathbb{C}_2 = \{x:\mathrm{T}, y:\mathrm{T}\cdot\mathrm{U}, z:\mathrm{U}\}$. The case of Example 3.6.11 demonstrates though that this would not work (per the stated intuitions): this would end up with a context of $\mathbb{C} = \{w:\mathrm{U}, x:\mathrm{T}, y:\mathrm{T}\cdot\mathrm{U}, z:\mathrm{T}\}$ (since y is the only common name) which if $\mathbb{C}_\epsilon = \{x\}$ would result in $\bigwedge_{x\in\mathbb{C}_\epsilon} \mathbb{C}_{\mathrm{T}}(x) = \mathrm{T}$ —* not *what is desired. In other words, the combination must be transitive and multiplication in the intersection only would lose this property.*

The cases for combining processes (summation and composition) deserve some thought, although the resultant rules are actually quite simple, albeit for different motivations. Both take the combined annotation as the meet of the two antecedent processes, in other words the lowest bound.

It is worth mentioning some of the possible alternatives for the summation rule to see why it takes the form it does. By definition a summation construct behaves as *either* P or Q, so in the absence of any knowledge as to *which* one will reduce, the more conservative approach is taken. An alternative that deserves consideration would be to require the annotations to match; that is:

$$\frac{\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : []^b; \mathbb{C}_1 \quad \Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{Q} : []^b; \mathbb{C}_2}{\Gamma, \Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} + \mathrm{Q} : []^b; \mathbb{C}_1, \mathbb{C}_2}$$

Note that by the definition of context union (Definition 3.6.3) if any names are common to both branches the process annotations are likely to be identical anyway, and this choice allows more processes to be typed.

The motivations behind the rule for composition are much less straightforward. The two antecedent processes are not linked in any obvious manner as is the case with summation; they may in fact be completely independent, even closed. The question arises then, what should the annotation (trustedness) on the *combination* be? Clearly it should have some bearing on their potential for interaction, both with themselves and with the external environment. If they are capable of interaction it is likely that they are already of the same trustedness, but if they are completely independent the issue is more complex: if they are isolated, one trusted and the other untrusted, which value is chosen for the overall result? Taking the greatest lower bound again (that is, multiplication; giving a result of untrusted) seems overly paranoid in the case where the untrusted process is in fact isolated from everything (for example, running in a sandbox). This appears to lead to the conclusion that the combined annotation should be calculated in a similar fashion to the Final rule, discussed earlier, and in fact earlier versions of this work (Hepburn and Wright 2003, 2004) mistakenly took this perspective.

The reason why this is mistaken is that it offers a means, early in a deduction, to effectively circumvent the other rules and upgrade the trustedness of a process. As a simple example, the process $x.\mathbf{0}^{\mathrm{U}}$ is considered untrusted and would have x in an untrusted context. However, the behaviourally-identical process $x.(\mathbf{0}^{\mathrm{U}}|\mathbf{0}^{\mathrm{T}})$ would be trusted and have x in a trusted context under the same rules. This situation appears too dangerous too ignore, and thus the compromise is that the composition rule takes the conservative approach, and the final rule will compute the trustedness of parallel processes with respect to the host.

There is a similar rationale behind the exclusion of a sub-typing rule for processes. It seems a reasonable intuition that a trusted process may safely be regarded as untrusted; however this makes the derivation of a judgement non-deterministic, and importantly makes it difficult or impossible to make guarantees of safety under the current structure of the contexts. As an example, with $x \in \mathbb{C}_\epsilon$, for any derivation of x.P with P as trusted and therefore placing x in a trusted context, there is a derivation with P as untrusted, and therefore x in an untrusted context, potentially altering the result after the final rule. For this reason such a rule is currently excluded and left for future work. The interaction between processes and input/output also mean that to preserve other safety results only processes of an identical type to that received may be transmitted.

Example 3.6.13 *Now consider the following two deductions:*

$$\begin{array}{ll}\Gamma_1 & \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \quad x.\mathbf{0}^{\mathrm{T}} : \mathsf{Proc}^{\mathrm{T}}; x : \mathrm{T} \\ \Gamma_2 & \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \quad y.\mathbf{0}^{\mathrm{U}} : \mathsf{Proc}^{\mathrm{U}}; y : \mathrm{U}\end{array}$$

Using the composition rule yields an untrusted process, as this is the meet of T *and* U*:*

$$\Gamma_1, \Gamma_2 \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} x.\mathbf{0}^{\mathrm{T}} \mid y.\mathbf{0}^{\mathrm{U}} : \mathsf{Proc}^{\mathrm{U}}; x : \mathrm{T}, y : \mathrm{U}$$

However, assuming $\mathbb{C}_\epsilon = \{x\}$ *(or indeed, contains neither* x *or* y*) then by the Final rule the resulting process is trusted overall*

$$\Gamma_1, \Gamma_2 \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} x.\mathbf{0}^{\mathrm{T}} \mid y.\mathbf{0}^{\mathrm{U}} : \mathsf{Proc}^{\mathrm{T}}$$

since $\bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}_{\mathrm{T}}(x) = \mathrm{T}$ *given* $\mathbb{C} = \{x : \mathrm{T}, y : \mathrm{U}\}$.

The most care needs to taken with regards to the communication (input and output) rules. Begin by examining the input rule:

$$\frac{\begin{array}{c}\Gamma_{\overrightarrow{V}}, x : (\overrightarrow{\sigma^c})^b, \overrightarrow{V} : \overrightarrow{\sigma^{c'}} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^d; \mathbb{C} \\ \overrightarrow{\sigma^c} \leq \overrightarrow{\sigma^{c'}} \quad \forall V \in \overrightarrow{V}.\mathbb{C}_{\mathrm{T}}(V) \geq_{\mathsf{B}} d, V \notin \mathbb{C}_\epsilon\end{array}}{\Gamma_{\overrightarrow{V}}, x : (\overrightarrow{\sigma^c})^b \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} x(\overrightarrow{V} : \overrightarrow{\sigma^{c'}}).\mathrm{P} : \mathsf{Proc}^d; \mathbb{C}_{\overrightarrow{V}}, x : d}$$

The consequent has the bound names removed from the environment and the context as expected, and the channel x added with a context of the base process. In the antecedent the types carried by x are required to match those of the parameters $\overrightarrow{V}$, modulo sub-typing. Note the direction of the sub-typing clause; this essentially says that a trusted input may be bound to an untrusted name (one of the core assumptions). Note that by the well-formed type rule, the annotations of all types input are multiplied by the annotation on x, so if x is untrusted then so are all variables that it carries (again, modulo sub-typing). There is an additional clause, $\forall V \in \overrightarrow{V}.\mathbb{C}_{\mathrm{T}}(V) \geq_{\mathsf{B}} d$ that is possibly somewhat unexpected. The purpose of this clause is related to the definition of the final rule; to prevent a name in an untrusted context of a trusted process becoming bound to a name capable of communication with the host (and thus making the process untrusted). The substitution lemma (page 111) has more detail on the need for this.

The output rule also needs careful attention:

$$\frac{\begin{array}{c}\Gamma, x : (c \cdot \overrightarrow{\sigma^d})^b \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^c; \mathbb{C}_1 \quad \overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma^{d'}}; \overrightarrow{\mathbb{C}_2} \\ \overrightarrow{\sigma^{d'}} \leq c \cdot \overrightarrow{\sigma^d} \quad \forall x \in \mathbb{C}_\epsilon.\overrightarrow{\mathbb{C}_{2\mathrm{T}}}(x) \geq_{\mathsf{B}} c \quad \forall \mathbb{C}_{2i}, V \in \mathrm{dom}\mathbb{C}_{2i}.\mathbb{C}_{2i}(V) \leq_{\mathsf{B}} d'_i\end{array}}{\Gamma, x : (c \cdot \overrightarrow{\sigma^d})^b, \overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \overline{x}[\overrightarrow{K}].\mathrm{P} : \mathsf{Proc}^c; \mathbb{C}_1, x : c, \overrightarrow{\mathbb{C}_2}, \mathsf{chans}\left(\overrightarrow{K}\right) : c}$$

Most of its components are expected; the types carried by the channel x must be multiplied by the annotation of the sending process, in addition to that of the channel itself as required by the well-formed type rule, thus untrusted processes may only send untrusted data. Note that a process may not appear on the left-hand side of a judgement, so separate judgements are required for the arguments. This need for separate judgements for the objects is the reason for allowing names to appear alone on the right-hand side of a judgement, in order to exploit the sequence shorthand notation. Once again the direction of the sub-type clause deserves note; it is the *reverse* of that in the input rule, in other words a trusted value may be sent along a channel that is typed as accepting untrusted inputs (note of course that at the destination — due to the type rules and the reverse clause in the input rule — it would be untrusted. This conforms to the requirements, stated above).

There are two clauses regarding constraints on the contexts of the objects that are not so transparent. The first, $\forall x \in \mathbb{C}_{\epsilon}.\mathbb{C}_{2\mathrm{T}}(x) \geq_{\mathsf{B}} c$, ensures an important property that processes typed as having a certain integrity will only have a more trusted annotation after the final rule, not a lower one. The second, $\forall \mathbb{C}_{2i}, V \in \mathrm{dom}\mathbb{C}_{2i}.\mathbb{C}_{2i}(V) \leq_{\mathsf{B}} d'_i$ concerns the nature of any processes that may be output (note that if K_i is a name then $\mathbb{C}_i$ will be the empty set). In particular, it enforces certain properties regarding agents output by processes regarded as untrusted; they must themselves contain untrusted names. This prevents a situation such as $x.\mathbf{0}^{\mathrm{T}}|\mathbf{0}^{\mathrm{U}}$ which would have the type $\mathsf{Proc}^{\mathrm{U}}$, but after being transmitted over an untrusted channel perhaps, then be able to communicate with trusted hosts. Note also that because the annotation d'_i must be multiplied by both the trustedness of P and of x then if either are untrusted d'_i also must be untrusted. Note that the same clause does not preclude a trusted process (using a trusted channel) from sending untrusted agents; just not those that may communicate with the host, as guaranteed by the first clause. In other words it is a guarantee that a process considered trustworthy will not send dangerous — with respect to the host —agents.

The context in the consequent is relatively unsurprising, being made up of the contexts from the base process P and the arguments ($\overrightarrow{K}$), the channel x mapping to the trustedness of the base process, and the channels of the arguments $\overrightarrow{K}$ all mapping to the trustedness of P, recalling that a judgement for a name does not introduce a context on its own.

The rules for abstraction and application are similar to input and output respectively; a natural result since the application of an abstraction to an argument can be *thought of* as an anonymous communication: for example

$(V : \sigma^c)\mathrm{P}\langle K\rangle$ can be imagined as being similar to

$$(\nu x : (\sigma^c)^{\mathrm{T}})(x(V : \sigma^c).\mathrm{P} \mid \overline{x}[K].\mathbf{0}^b)$$

where b is also the trustedness of P, and the anonymous channel is trusted so the type $(\sigma^c)^{\mathrm{T}}$ is well-formed for all σ^c. In essence it is forming a composition of an input and an output-prefixed process, without the composition explicitly involved, so the same principles apply (including the sub-typing clause).

The certify rule by contrast — and perhaps surprisingly given its power — is remarkably simple (at least in comparison).

$$\frac{\begin{array}{l}\Gamma_V, V : \sigma^{\mathrm{T}}, x : (\sigma^i)^e \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^c; \mathbb{C}_1 \quad \mathbb{C}_{1\mathrm{T}}(V) \geq_{\mathsf{B}} c \\ \Theta_V, V : \sigma^{\mathrm{U}}, x : (\sigma^i)^e \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}<} \mathrm{Q} : \mathsf{Proc}^d; \mathbb{C}_2 \quad \mathbb{C}_{2\mathrm{T}}(V) \geq_{\mathsf{B}} d \quad V \notin \mathbb{C}_\epsilon\end{array}}{\begin{array}{r}\Gamma_V, \Theta_V, x : (\sigma^i)^e \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} x(V : \sigma^i)_{\mathrm{certify}}\mathrm{P} \oplus \mathrm{Q} : \mathsf{Proc}^{i\cdot c+\bar{i}\cdot d}; \\ \mathbb{C}_{1V} \langle i\rangle \mathbb{C}_{2V}, x : (i \cdot c + \bar{i} \cdot d)\end{array}}$$

The first thing that should be noted is that it is essentially an input construct, so is defined similarly. Secondly though, note that there is no sub-typing clause on the bound variable involved: this is because the definition of sub-typing (Definition 3.4.3) only considers the outermost annotation of the type, and this annotation is here *defined by the rule itself*. That is, the input parameter is bound to a trusted variable in one branch (P), and an untrusted variable (with an identical base-type) in the other branch. Note that while the type of the bound variable(s) differs from that received from x it is *not* sub-typing: the type of the parameter bound to the appropriate variable will match that of the variable exactly *at run-time* due to run-time coercion. The bound variable (V) is, as expected, removed from the environment and context in the consequent. There are also similar clauses on the context of the bound variable in each branch not being any less safe than the trustedness of the branch.

An interesting point occurs when deciding upon the annotation of the entire construct: two processes are being combined, as in a summation or composition construct, however this time it is *not* the case that there is no knowledge as to how they might be used. Whereas a summation construct could reduce to either branch (much like a certify form), and a parallel composition might have independent or intertwined antecedents, with certify one extra bit of information graces the deduction: the annotation on the input variable (here represented as i)! It is thus *known* (thanks to the oracle-nature of the verification procedure used) that one branch will execute if i turns out to be trusted, and the other if its integrity is found to be lacking. Armed

with this knowledge, the following annotation expression for the entire combination may be constructed: $i \cdot c + \bar{i} \cdot d$. It is not hard to verify that if $i = \mathrm{T}$ then the expression reduces to c (the trustedness of the first alternative P), and to d if $i = \mathrm{U}$. (This construct has been independently proposed elsewhere in different circumstances; see Wright (1992, Section 6.1.4) and Wright (1991, Section 6.2)). This is of course the same reasoning that was utilised in the certify rule for System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$, applied to process trustedness instead of security levels.

The final context in a certify expression is also worth examining. The channel x appears under the context of the overall process $(i \cdot c + \bar{i} \cdot d)$ just as in an input or output form. When combining the contexts of the two branches ($\mathbb{C}_1$ and $\mathbb{C}_2$) there is again a little more flexibility than simply requiring the images of any names in the intersection of the domains match; see Definition 3.6.3 for the details of how the same expression is used to construct a new context ($\mathbb{C}_{1V} \langle i \rangle \mathbb{C}_{2V}$, in a similar fashion to the annotation).

3.6.3 Typed Labelled-Transition-Semantics

The typed transition semantics are extended in the logical way. They are more or less the same as those for the vanilla system that appeared earlier in Section 2.3.3, with the types altered as necessary and additional cases for certify reduction. They appear in Figures 3.9 and 3.10, with the simpler (predominantly structural) cases presented first, followed by the significantly more complicated cases involving the base communication cases. The number of side-conditions imposed on the base cases appears over-whelming, but correspond directly to the corresponding conditions in the type rules. Note that input and output (and certify, with a few extra cases dealing with coercion) are nearly identical, with respect to symmetry in the consequent. This is to be expected, as an input involves a corresponding output in the anonymous process, and vica-versa.

The communication cases all encode the subset of names transferred between environments, and indicate ($\exists \mathbb{C}_1''' \subseteq \mathbb{C}_1''$) that the contexts transfered may be a subset if some arguments do not bind anything (technically this is true of the environments as well, but the weaken rule ensures that an extra name inadvertently added to an environment doesn't affect the typing).

3.6.4 Comparison between Security Levels and Process Trustedness

The reader may have noticed that the form and treatment of the process annotations in the rules of System $\mathcal{T}_{\mathsf{HO}\pi\leq}$ are very similar to the security levels

$$\frac{\langle \Theta | \mathbb{C}_2, \Gamma | \mathbb{C}_1, \mathrm{P} \rangle \xrightarrow{\mu} \langle \Theta' | \mathbb{C}'_2, \Gamma' | \mathbb{C}'_1, \mathrm{P}' \rangle \Vdash_{\text{certify}} \mathbb{R} \quad \mathrm{P} \equiv_\alpha \mathrm{Q}}{\langle \Theta | \mathbb{C}_2, \Gamma | \mathbb{C}_1, \mathrm{Q} \rangle \xrightarrow{\mu} \langle \Theta' | \mathbb{C}'_2, \Gamma' | \mathbb{C}'_1, \mathrm{P}' \rangle \Vdash_{\text{certify}} \mathbb{R}} \ (alp.)$$

$$\frac{\langle \Theta | \mathbb{C}_2, \Gamma | \mathbb{C}_1, \mathrm{P}\{\overrightarrow{K}/\overrightarrow{V}\} \rangle \xrightarrow{\mu} \langle \Theta' | \mathbb{C}'_2, \Gamma' | \mathbb{C}'_1, \mathrm{P}' \rangle \Vdash_{\text{certify}} \mathbb{R}}{\langle \Theta | \mathbb{C}_2, \Gamma | \mathbb{C}_1, (\overrightarrow{V})\mathrm{P}\langle\overrightarrow{K}\rangle \rangle \xrightarrow{\mu} \langle \Theta' | \mathbb{C}'_2, \Gamma' | \mathbb{C}'_1, \mathrm{P}' \rangle \Vdash_{\text{certify}} \mathbb{R}} \ (app.)$$

$$\frac{\langle \Theta | \mathbb{C}_2, \Gamma | \mathbb{C}_1, \mathrm{P} \rangle \xrightarrow{\mu} \langle \Theta' | \mathbb{C}'_2, \Gamma' | \mathbb{C}'_1, \mathrm{P}' \rangle \Vdash_{\text{certify}} \mathbb{R}}{\langle \Theta | \mathbb{C}_2, \Gamma | \mathbb{C}_1 \cup \mathbb{C}''_1, \mathrm{P} + \mathrm{Q} \rangle \xrightarrow{\mu} \langle \Theta' | \mathbb{C}'_2, \Gamma' | \mathbb{C}'_1, \mathrm{P}' \rangle \Vdash_{\text{certify}} \mathbb{R}} \ (sum.)$$

$$\frac{\langle \Theta | \mathbb{C}_2, \Gamma | \mathbb{C}_1, \mathrm{P} \rangle \xrightarrow{\mu} \langle \Theta' | \mathbb{C}'_2, \Gamma' | \mathbb{C}'_1, \mathrm{P}' \rangle \Vdash_{\text{certify}} \mathbb{R} \quad BN(\mu) \cap FN(\mathrm{Q}) = \emptyset}{\langle \Theta | \mathbb{C}_2, \Gamma | \mathbb{C}_1 \cup \mathbb{C}''_1, \mathrm{P} | \mathrm{Q} \rangle \xrightarrow{\mu} \langle \Theta' | \mathbb{C}'_2, \Gamma' | \mathbb{C}'_1 \cup \mathbb{C}''_1, \mathrm{P}' | \mathbb{R}(\mathrm{Q}) \rangle \Vdash_{\text{certify}} \mathbb{R}} \ (comp.)$$

$$\frac{\langle \Theta | \mathbb{C}_2, \Gamma | \mathbb{C}_1, \mathrm{P} \rangle \xrightarrow{\mu} \langle \Theta' | \mathbb{C}'_2, \Gamma' | \mathbb{C}'_1, \mathrm{P}' \rangle \Vdash_{\text{certify}} \mathbb{R}}{\langle \Theta | \mathbb{C}_2, \Gamma | \mathbb{C}_1, !\mathrm{P} \rangle \xrightarrow{\mu} \langle \Theta' | \mathbb{C}'_2, \Gamma' | \mathbb{C}'_1 \cup \mathbb{C}_1, \mathrm{P}' | \mathbb{R}(!\mathrm{P}) \rangle \Vdash_{\text{certify}} \mathbb{R}} \ (rep.)$$

$$\frac{\begin{array}{c}\langle \Theta | \mathbb{C}_2, \Gamma_x \cup \{x : \sigma^b\} | \mathbb{C}_1, \mathrm{P} \rangle \xrightarrow{\mu} \langle \Theta' | \mathbb{C}'_2, \Gamma'_x \cup \{x : \sigma^b\} | \mathbb{C}'_1, \mathrm{P}' \rangle \Vdash_{\text{certify}} \mathbb{R} \\ x \notin FN(\mu) \cup BN(\mu) \quad x \notin \mathrm{dom}\mathbb{C}_2\end{array}}{\langle \Theta_x | \mathbb{C}_2, \Gamma_x | \mathbb{C}_{1x}, (\nu x : \sigma^b)\mathrm{P} \rangle \xrightarrow{\mu} \langle \Theta'_x | \mathbb{C}'_2, \Gamma'_x | \mathbb{C}'_{1x}, (\nu x : \sigma^b)\mathrm{P}' \rangle \Vdash_{\text{certify}} \mathbb{R}} \ (res.)$$

$$\frac{\begin{array}{c}\langle \Theta | \mathbb{C}_2, \Gamma_w \cup \{w : \sigma^b\} | \mathbb{C}_1, \mathrm{P} \rangle \xrightarrow{(\nu \overrightarrow{z})\overline{x}[\overrightarrow{K}]} \\ \langle \Theta' | \mathbb{C}'_2, \Gamma'_w \cup \{w : \sigma^b\} | \mathbb{C}'_1, \mathrm{P}' \rangle \Vdash_{\text{certify}} \mathbb{R} \\ w \neq x \quad w \in FN\left(\overrightarrow{K}\right) - \overrightarrow{z} \quad w \notin \mathrm{dom}\mathbb{C}_2\end{array}}{\begin{array}{l}\langle \Theta_w | \mathbb{C}_2, \Gamma_w | \mathbb{C}_{1w}, (\nu w : \sigma^b)\mathrm{P} \rangle \xrightarrow{(\nu w \overrightarrow{z})\overline{x}[\overrightarrow{K}]} \\ \qquad \langle \Theta'_w | \mathbb{C}'_2, \Gamma'_w | \mathbb{C}'_{1w}, (\nu w : \sigma^b)\mathrm{P}' \rangle \Vdash_{\text{certify}} \mathbb{R}\end{array}} \ (open)$$

Figure 3.9: Typed Labelled Transition Semantics for System $\mathcal{T}_{\mathsf{HO}\pi\leq}$

$$\frac{\begin{array}{c}\overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma^d}; \overrightarrow{\mathbb{C}'_1} \quad \bigcup \overrightarrow{\Gamma'} \subseteq \Gamma \quad \mathrm{dom} \bigcup \overrightarrow{\Gamma'} = FN\left(\overrightarrow{K}\right) \\ \forall x \in \mathbb{C}_\epsilon . \overrightarrow{\mathbb{C}'_{1\mathrm{T}}}(x) \geq_{\mathsf{B}} b \quad \forall \mathbb{C}'_{1i}, V \in \mathrm{dom}\mathbb{C}'_{1i} . \mathbb{C}'_{1i}(V) \leq_{\mathsf{B}} d_i \\ \mathbb{C}''_1 = \overrightarrow{\mathbb{C}'_1}, \mathsf{chans}\left(\overrightarrow{K}\right) : b \quad \exists \mathbb{C}'''_1 \subseteq \mathbb{C}''_1 \end{array}}{\begin{array}{l}\langle \Theta | \mathbb{C}_2, \Gamma | \mathbb{C}_1 \cup \mathbb{C}''_1 \cup \{x : b\}, \overline{x}[\overrightarrow{K}].\mathrm{P}\rangle \xrightarrow{\overline{x}[\overrightarrow{K}]} \\ \qquad \langle \Theta \cup \overrightarrow{\Gamma'} | \mathbb{C}_2 \cup \mathbb{C}'''_1, \Gamma | \mathbb{C}_1, \mathrm{P}\rangle \Vdash_{\mathrm{certify}} \mathsf{Id}\end{array}} \ (out.)$$

$$\frac{\begin{array}{c}\overrightarrow{\Theta'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma^{d'}}; \overrightarrow{\mathbb{C}'_2} \quad \bigcup \overrightarrow{\Theta'} \subseteq \Theta \quad \mathrm{dom} \bigcup \overrightarrow{\Theta'} = FN\left(\overrightarrow{K}\right) \\ \forall x \in \mathbb{C}_\epsilon . \overrightarrow{\mathbb{C}'_{2\mathrm{T}}}(x) \geq_{\mathsf{B}} b \quad \forall \mathbb{C}'_{2i}, V \in \mathrm{dom}\mathbb{C}'_{2i} . \mathbb{C}'_{2i}(V) \leq_{\mathsf{B}} d'_i \\ \overrightarrow{\sigma^{d'}} \leq \overrightarrow{\sigma^d} \quad \mathbb{C}''_2 = \overrightarrow{\mathbb{C}'_2}, \mathsf{chans}\left(\overrightarrow{K}\right) : b \quad \exists \mathbb{C}'''_2 \subseteq \mathbb{C}''_2 \end{array}}{\begin{array}{l}\langle \Theta | \mathbb{C}_2 \cup \mathbb{C}''_2, \Gamma | \mathbb{C}_1 \cup \{x : b\}, x(\overrightarrow{V} : \overrightarrow{\sigma^d}).\mathrm{P}\rangle \xrightarrow{x(\overrightarrow{K})} \\ \qquad \langle \Theta | \mathbb{C}_2, \Gamma \cup \overrightarrow{\Theta'} | \mathbb{C}_1 \cup \mathbb{C}'''_2, \mathrm{P}\{\overrightarrow{K}/\overrightarrow{V}\}\rangle \Vdash_{\mathrm{certify}} \mathsf{Id}\end{array}} \ (inp.)$$

$$\frac{\begin{array}{c}\langle \Theta | \mathbb{C}_2, \Gamma | \mathbb{C}_1, \mathrm{P}\rangle \xrightarrow{(\nu \overrightarrow{z})\overline{x}[\overrightarrow{K}]} \langle \Theta' | \mathbb{C}'_2, \Gamma' | \mathbb{C}'_1, \mathrm{P}'\rangle \Vdash_{\mathrm{certify}} \mathsf{Id} \\ \langle \Gamma | \mathbb{C}_1, \Theta | \mathbb{C}_2, \mathrm{Q}\rangle \xrightarrow{x(\overrightarrow{K})} \langle \Gamma' | \mathbb{C}'_1, \Theta' | \mathbb{C}'_2, \mathrm{Q}\rangle \Vdash_{\mathrm{certify}} \mathbb{R} \quad \overrightarrow{z} \notin FN(\mathrm{Q}) \end{array}}{\begin{array}{l}\langle \Delta | \mathbb{C}_3, \Gamma \cup \Theta | \mathbb{C}_1 \cup \mathbb{C}_2 \mathrm{P} | \mathrm{Q}\rangle \xrightarrow{\tau} \\ \qquad \langle \mathbb{R}(\Delta | \mathbb{C}_3), \mathbb{R}(\Gamma' \cup \Theta' | \mathbb{C}'_1 \cup \mathbb{C}'_2), \mathbb{R}(\mathrm{P}' | \mathrm{Q}')\rangle \Vdash_{\mathrm{certify}} \mathbb{R}\end{array}} \ (comm.)$$

$$\frac{\begin{array}{c}\Gamma(x) = (\sigma^i)^e \quad \mathrm{certify}(K) = \mathrm{T} \quad \mathbb{R} = \mathsf{Id}[i := \mathrm{T}] \\ \Theta' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} K : \sigma^f; \mathbb{C}'_2 \quad \Theta' \subseteq \Theta \quad \mathrm{dom}\Theta' = FN(K) \\ \forall V \in \mathrm{dom}\mathbb{C}'_2 . \mathbb{C}'_2(V) \leq_{\mathsf{B}} f \quad \forall x \in \mathbb{C}_\epsilon . \mathbb{C}'_{2\mathrm{T}}(x) \geq_{\mathsf{B}} b = i \cdot c + \overline{i} \cdot d \\ \sigma^f \leq \sigma^i \quad \mathbb{C}''_2 = \mathbb{C}'_2, \mathsf{chans}(K) : b \quad \exists \mathbb{C}'''_2 \subseteq \mathbb{C}''_2 \end{array}}{\begin{array}{l}\langle \Theta | \mathbb{C}_2 \cup \mathbb{C}''_2, \Gamma | \mathbb{C}'_1 \ \langle i \rangle \ \mathbb{C}''_1 \cup \{x : b\}, x(V : \sigma^i)_{\mathrm{certify}}?\mathrm{P} \oplus \mathrm{Q}\rangle \xrightarrow{x(K)} \\ \qquad \langle \mathbb{R}(\Theta | \mathbb{C}_2), \mathbb{R}(\Gamma \cup \Theta' | \mathbb{C}'_1 \cup \mathbb{C}'''_2), \mathbb{R}(\mathrm{P}\{K/V\})\rangle \Vdash_{\mathrm{certify}} \mathbb{R}\end{array}} \ (cert - T)$$

$$\frac{\begin{array}{c}\Gamma(x) = (\sigma^i)^e \quad \mathrm{certify}(K) = \mathrm{U} \quad \mathbb{R} = \mathsf{Id}[i := \mathrm{U}] \\ \Theta' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} K : \sigma^f; \mathbb{C}'_2 \quad \Theta' \subseteq \Theta \quad \mathrm{dom}\Theta' = FN(K) \\ \forall V \in \mathrm{dom}\mathbb{C}'_2 . \mathbb{C}'_2(V) \leq_{\mathsf{B}} f \quad \forall x \in \mathbb{C}_\epsilon . \mathbb{C}'_{2\mathrm{T}}(x) \geq_{\mathsf{B}} b = i \cdot c + \overline{i} \cdot d \\ \sigma^f \leq \sigma^i \quad \mathbb{C}''_2 = \mathbb{C}'_2, \mathsf{chans}(K) : b \quad \exists \mathbb{C}'''_2 \subseteq \mathbb{C}''_2 \end{array}}{\begin{array}{l}\langle \Theta | \mathbb{C}_2 \cup \mathbb{C}''_2, \Gamma | \mathbb{C}'_1 \ \langle i \rangle \ \mathbb{C}''_1 \cup \{x : b\}, x(V : \sigma^i)_{\mathrm{certify}}?\mathrm{P} \oplus \mathrm{Q}\rangle \xrightarrow{x(K)} \\ \qquad \langle \mathbb{R}(\Theta | \mathbb{C}_2), \mathbb{R}(\Gamma \cup \Theta' | \mathbb{C}''_1 \cup \mathbb{C}'''_2), \mathbb{R}(\mathrm{Q}\{K/V\})\rangle \Vdash_{\mathrm{certify}} \mathbb{R}\end{array}} \ (cert - U)$$

Figure 3.10: Typed Labelled Transition Semantics for System $\mathcal{T}_{\mathsf{HO}\pi\leq}$(continued)

in the rules of System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$. The differences are fairly subtle; there is no notion of process identity in the first-order system, so assigning a trustedness level would not make sense. The main difference is that the security levels are channel-oriented, whereas the process annotations are process-oriented. That is, a security level states that all channels used must be at least as safe as the security level, and thus ensures that a process typed at a high security level cannot be affected by channels of a lower integrity.

The process annotations have a different focus. Firstly, since processes can be transmitted it is logical that they may become corrupted, and hence should have an integrity level associated. They are an experiment to see if security can be enforced on a process-level, rather than a channel level: the intent is that processes should be able to use untrusted channels to communicate, but that untrusted processes should not be able to affect trusted processes. A possible change to the rules would be to include similar conditions to those in System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$; such that a trusted process can only use trusted channels. This view seems overly restrictive, and would detract from the current experiment.

3.7 Discussion

In this Chapter four related type systems were presented to implement the ideas of this thesis:

- System $\mathcal{T}_{\mathsf{FO}\pi}$: A system of type annotations enforces the separation of trusted and untrusted data, even in the presence of dynamic topologies. The certify addition to the language enables safe coercion of integrity levels, based on the results of run-time verification checks.

- System $\mathcal{T}_{\mathsf{FO}\pi\leq}$: An extension of the previous system that allows a form of sub-typing based on channel annotations. This admits the possibility of write-down (trusted names may be written to untrusted channels), but the type system prevents the reverse from occurring.

- System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$: A further extension that incorporates a security level on a derivation, to guarantee that channels of an integrity lower than this level cannot affect a reduction.

- System $\mathcal{T}_{\mathsf{HO}\pi\leq}$: A system, incorporating a similar type system and certify construct, built on the higher-order π-calculus to enable easier reasoning about mobile code.

For the remainder of this dissertation no mention will be made of System $\mathcal{T}_{\mathsf{FO}\pi\leq}$, as it was primarily used as a development vehicle to introduce sub-typing prior to incorporating security levels as well.

3.7.1 Relation to Existing Work

It is useful before proceeding to examine some of the work mentioned in Chapter 1 again in light of the systems presented here.

The closest in approach to safe coercion was the work on Robust Declassification by Myers et al. (2004). This obviously has many common viewpoints to the work presented in this thesis: the main points of difference appear to be the implications for the programmer, and that their work is primarily a static analysis with a reduced emphasis on run-time checks. While there is support for run-time checks in Myers and Liskov (1997), its usage in a secrecy setting is tricky as the failure of a check can still leak information, in a similar manner to that of a failed password check. From a programmer's perspective it is possible that an operator with explicit reference to a verification procedure might be easier to understand than operators such as `declassify/endorse` which appear to require more of an understanding of the underlying information flow, but this is perhaps a difficult comparison to make give the slightly different perspectives of each system.

While the systems $\mathcal{T}_{\mathsf{FO}\pi}$, $\mathcal{T}_{\mathsf{FO}'\pi\leq}$, and $\mathcal{T}_{\mathsf{HO}\pi\leq}$ have no concept of a region explicitly, as contained in Confined-λ (Kirli 2001), it would appear that the execution contexts of system $\mathcal{T}_{\mathsf{HO}\pi\leq}$ have a similar role. If trustedness levels of processes are seen as regions (with untrusted contained in trusted), then contexts enforce the property that no process will communicate (modulo a prior use of the sub-type rule) with a process of a different trustedness level.

A similar by more fine-grained approach (to that of Confined-λ) was offered by Yoshida and Hennessy (2000), using interfaces restricting the capabilities of names, and in an extension using dependent types to type abstractions. This is a very interesting, and very powerful, idea. System $\mathcal{T}_{\mathsf{HO}\pi\leq}$ takes a different approach, again based on the execution contexts, in which rather than rejecting an application of a process to another based on interface, the formation of the abstraction (and an input) is constrained based on its trustedness and contextual information, and the external context. That is, in the abstraction rule:

$$\frac{\Gamma_{\overrightarrow{V}}, \overrightarrow{V} : \overrightarrow{\sigma^c} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C} \quad \forall V \in \overrightarrow{V}.\mathbb{C}_{\mathrm{T}}(V) \geq_{\mathsf{B}} b, V \notin \mathbb{C}_\epsilon}{\Gamma_{\overrightarrow{V}} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} (\overrightarrow{V} : \overrightarrow{\sigma^c})\mathrm{P} : (\overrightarrow{\sigma^c}) \rightarrow \mathsf{Proc}^b; \mathbb{C}_{\overrightarrow{V}}}$$

the clause $\forall V \in \overrightarrow{V}.\mathbb{C}_{\mathrm{T}}(V) \geq_{\mathsf{B}} b$ requires that the context of any name

bound is at least as safe as the trustedness of the process being abstracted; this, together with a complementary clause $\forall x \in \mathbb{C}_\epsilon.\overrightarrow{\mathbb{C}_{2\mathrm{T}}}(x) \geq_{\mathsf{B}} b$ regarding the context of names in the "argument" to the abstraction ensures that an application will not decrease in integrity. In some regard therefore, a context may be regarded as interface for a process, however it is not as fine-grained as that in Yoshida and Hennessy (2000).

An extension to the work of Yoshida and Hennessy (2000) was presented by Vivas and Yoshida (2002), using dynamic filters. The filter operator clearly resembles a localised and dynamic version of $\mathbb{C}_\epsilon$ (or alternatively even a simple version of certify, with predefined behaviour per name and no branching capability), and as such this work, along with that on declassification, probably bears the closest resemblance to that of this thesis. It would be interesting to see what other comparisons may be drawn, and to see if a union might be made between the screening operator and certify: for example, it would be interesting to add trustedness information to the filters (so that for example a reduction may only occur if the object of the action exists in the filter with the correct I/O usage, and is at least as safe as the specified trustedness). Whether coercion can fit in the framework is another intriguing question.

Chapter 4

Safety of the Type Systems

There are two components to demonstrating the safety of the systems: a demonstration of type soundness, in other words that types are preserved under reduction, and a security property. The first is shown by subject reduction, and as will be seen this is not as straight-forward as might be expected, due to the presence of run-time coercion. The security property for the first-order system is a guarantee of non-interference: essentially, if a system is well-typed then no low-integrity name can interfere — by changing the behaviour — of a process typed at a higher level. For the higher-order systems an alternative perspective based on observability defined by the security policy is adopted. If a process is typed as trusted after the final rule, it should never be able to evolve such that an untrusted sub-process is capable of communication (that is, is guarded with one of the names in the external context) with the host.

Subject reduction is presented in two stages; firstly a theorem showing that a single-step reduction preserves typing, and secondly a corollary demonstrating that the corresponding multi-step reduction is similarly safe.

Results are presented for only three out of the four systems presented in Chapter 3. System $\mathcal{T}_{\mathsf{FO}\pi}$ only has type-soundness demonstrated, as it is not a security system. Results for System $\mathcal{T}_{\mathsf{FO}\pi\leq}$ are not included here, as it is not a security system and its soundness results are contained in System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$ (the reader is referred to Lemma 3.5.2 on Page 62). Full results for both type soundness and security are described for Systems $\mathcal{T}_{\mathsf{FO}'\pi\leq}$ and $\mathcal{T}_{\mathsf{HO}\pi\leq}$.

4.1 Safety of System $\mathcal{T}_{\mathsf{FO}\pi}$

As mentioned previously, soundness of the type systems is proven by a demonstration that types are preserved under reductions. This is achieved

in two parts; a substitution lemma, and the subject reduction theorem itself.

In this Section and those that follow, the result statements — and where applicable, the proofs — will ignore the conversion type rule, as it has no affect on any part of the environment, etc. This is purely for reasons of clarity in the statements, to avoid having to repeat each result for the two different forms of deduction.

4.1.1 Substitution Lemma

The substitution lemma for system $\mathcal{T}_{\mathsf{FO}\pi}$ is quite standard; it states that a well-formed process remains well-formed when a variable is replaced with another of a similar type.

Lemma 4.1.1 (Substitution Lemma) *If there is a valid deduction*

$$\Gamma_{xy}, x : \sigma^b \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}$$

in System $\mathcal{T}_{\mathsf{FO}\pi}$, then there is also a valid deduction for

$$\Gamma_{xy}, y : \sigma^b \vdash_{\mathrm{FO}} \mathrm{P}\,\{y/x\} : \mathsf{Proc}$$

Proof. By induction on the derivation of $\Gamma_{xy}, x : \sigma^b \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}$.

- Zero: The axiom as it appears in the rules

  $$\frac{}{\emptyset \vdash_{(\mathrm{FO})} \mathbf{0} : \mathsf{Proc}}$$

  is not in the form required (with the domain of the environment containing x), however if the environment is extended using the weaken rule it may be noted that by Definition 2.2.6 $\mathbf{0}\{y/x\} = \mathbf{0}$ for every x and y as required.

- Weaken: the rule has the form

  $$\frac{\Gamma \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc} \quad z \notin \mathrm{dom}\Gamma \quad \vdash \sigma'^c}{\Gamma, z : \sigma'^c \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}}$$

  Assuming $\Gamma = \Gamma'_{xy}, x : \sigma^b$ for some Γ', then by the induction hypothesis $\Gamma'_{xy}, y : \sigma^b \vdash_{\mathrm{FO}} \mathrm{P}\{y/x\} : \mathsf{Proc}$ is valid. Hence by the weaken rule $\Gamma'_{xy}, y : \sigma^b, z : \sigma'^c \vdash_{\mathrm{FO}} \mathrm{P}\{y/x\} : \mathsf{Proc}$ is also valid, noting that $z \neq x, y$ by the clause in the weaken rule.

- Replication: by the induction hypothesis on the antecedent and the replication rule.

- Summation: the rule has the form

$$\frac{\Gamma \vdash_{(\mathrm{FO})} \mathrm{P} : \mathsf{Proc} \quad \Theta \vdash_{(\mathrm{FO})} \mathrm{Q} : \mathsf{Proc}}{\Gamma, \Theta \vdash_{(\mathrm{FO})} \mathrm{P} + \mathrm{Q} : \mathsf{Proc}}$$

  By the induction hypothesis the result holds for both the antecedents individually, that is

$$\Gamma'_{xy}, y : \sigma^b \vdash_{\mathrm{FO}} \mathrm{P}\{y/x\} : \mathsf{Proc} \quad \text{and} \quad \Theta'_{xy}, y : \sigma^b \vdash_{\mathrm{FO}} \mathrm{Q}\{y/x\} : \mathsf{Proc}$$

  where $\Gamma = \Gamma'_{xy}, x : \sigma^b$ and $\Theta = \Theta'_{xy}, x : \sigma^b$ for some Γ' and Θ'. Hence by the summation rule and the definition of substitution,

$$\Gamma'_{xy}, \Theta'_{xy}, y : \sigma^b \vdash_{\mathrm{FO}} (\mathrm{P} + \mathrm{Q})\{y/x\} : \mathsf{Proc}$$

  as required.

- Composition: similarly.

- Restriction: by the induction hypothesis, noting that due to the assumption of Barendregt's Convention (Page 21) and the statement which has y free in the consequent the case where x or y is the restricted name need not be considered.

- Conversion: trivial, by the induction hypothesis.

- Output: The rule has the form

$$\frac{\Gamma, w : (\overrightarrow{\sigma'^d})^c, \overrightarrow{z} : \overrightarrow{\sigma'^d} \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}}{\Gamma, w : (\overrightarrow{\sigma'^d})^c, \overrightarrow{z} : \overrightarrow{\sigma'^d} \vdash_{(\mathrm{FO})} \overline{w}[\overrightarrow{z}].\mathrm{P} : \mathsf{Proc}}$$

  By the induction hypothesis the result holds for the antecedent, in other words assuming $\Gamma = \Gamma'_{xy}, x : \sigma^b$ then $\Gamma'_{xy}, y : \sigma^b, w : (\overrightarrow{\sigma'^d})^c, \overrightarrow{z} : \overrightarrow{\sigma'^d} \vdash_{\mathrm{FO}} \mathrm{P}\{y/x\} : \mathsf{Proc}$ is valid. There are three cases to consider in the consequent; x and y distinct from w and $\overrightarrow{z}$, $x = w$, and $x \in \overrightarrow{z}$ (note that due to the lack of recursive types the case where both $x = w$ and $x \in \overrightarrow{z}$ cannot occur). The first holds easily by the output rule. The second case follows the use of induction hypothesis to give $\Gamma'_{xy}, y : (\overrightarrow{\sigma'^d})^c, \overrightarrow{z} : \overrightarrow{\sigma'^d} \vdash_{\mathrm{FO}} \overline{y}[\overrightarrow{z}].\mathrm{P}\{y/x\} : \mathsf{Proc}$, recalling from Definition 2.2.6 that $(\overline{x}[\overrightarrow{z}].\mathrm{P})\{y/x\} = \overline{y}[\overrightarrow{z}].(\mathrm{P}\{y/x\})\{y/x\}$ when $x \notin \overrightarrow{z}$. The third case follows similarly.

- Input: Similarly (noting that by the statement x or y cannot appear in the bound input names).

- Certify: Similarly, noting that the bound name cannot be x or y.

□

Corollary 4.1.2 (Substitution Corollary) *If there is a valid deduction*

$$\Gamma_{\overrightarrow{xy}}, \overrightarrow{x} : \overrightarrow{\sigma^b} \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}$$

in System $\mathcal{T}_{\mathsf{FO}\pi}$, where all $\overrightarrow{x}$ are distinct, then

$$\Gamma_{\overrightarrow{xy}}, \overrightarrow{y} : \overrightarrow{\sigma^b} \vdash_{\mathrm{FO}} \mathrm{P}\,\{\overrightarrow{y}/\overrightarrow{x}\} : \mathsf{Proc}$$

is also a valid deduction.

Proof. By Lemma 4.1.1 (the Substitution Lemma). □

4.1.2 Subject Reduction

The subject reduction theorem for system $\mathcal{T}_{\mathsf{FO}\pi}$ is also moderately familiar, with one addition. Because it is possible for coercion to occur during a reduction (that involves certify), the environment in the consequent may not be *identical* to that before the reduction (as is commonly the case in subject reduction). This quandary is solved by making the environment in the consequent the same, modulo a substitution on annotations. This substitution is precisely defined by the reduction path, and represents exactly each coercion that occurred as a result of certify. It is exactly that defined by the typed labelled transition semantics of Figure 3.3.

The majority of the proof is contained in a lemma over this definition, before the main result is presented.

Lemma 4.1.3 *If there is a valid deduction $\Gamma \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}$ in System $\mathcal{T}_{\mathsf{FO}\pi}$, and a reduction such that*

$$\langle \Theta, \Gamma, \mathrm{P} \rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}' \rangle \Vdash_{\mathrm{certify}} \mathbb{R}$$

then there is a valid deduction

$$\mathbb{R}(\Gamma \cup \Theta) \vdash_{\mathrm{FO}} \mathbb{R}(\mathrm{P}') : \mathsf{Proc}$$

Note that both environments are used in the second reduction, to incorporate any names acquired (such as via an input) from the anonymous process.

Proof. By induction on the derivation of $\langle \Theta, \Gamma, \mathrm{P} \rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}' \rangle \Vdash_{\mathrm{certify}} \mathbb{R}$ and on the deduction $\Gamma \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}$.

- Output: The rule states

  $$\langle \Theta, \Gamma \cup \{\overrightarrow{y} : \overrightarrow{\sigma^b}\}, \overline{x}[\overrightarrow{y}].\mathrm{P}\rangle \xrightarrow{\overline{x}[\overrightarrow{y}]} \langle \Theta \cup \{\overrightarrow{y} : \overrightarrow{\sigma^b}\}, \Gamma \cup \{\overrightarrow{y} : \overrightarrow{\sigma^b}\}, \mathrm{P}\rangle \Vdash_{\text{certify}} \mathsf{Id}$$

  and from the statement $\Gamma, \overrightarrow{y} : \overrightarrow{\sigma^b} \vdash_{(\mathrm{FO})} \overline{x}[\overrightarrow{y}].\mathrm{P} : \mathsf{Proc}$. This deduction must have ended in zero or more uses of the weaken rule, preceded by an application of the output type rule with antecedent

  $$\Gamma'', x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma^b} \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}$$

  (where $\Gamma'', x : (\overrightarrow{\sigma^b})^c$ is a subset of Γ). Then by as many applications of the weaken rule as necessary,

  $$\Theta, \Gamma, \overrightarrow{y} : \overrightarrow{\sigma^b} \vdash_{(\mathrm{FO})} \mathrm{P} : \mathsf{Proc}$$

  as required (observing that the substitution generated is the identity substitution).

- Input: The rule states

  $$\langle \Theta \cup \{\overrightarrow{y} : \overrightarrow{\sigma^b}\}, \Gamma, x(\overrightarrow{z} : \overrightarrow{\sigma^b}).\mathrm{P}\rangle \xrightarrow{x(\overrightarrow{y})} \langle \Theta \cup \{\overrightarrow{y} : \overrightarrow{\sigma^b}\}, \Gamma \cup \{\overrightarrow{y} : \overrightarrow{\sigma^b}\}, \mathrm{P}\{\overrightarrow{y}/\overrightarrow{z}\}\rangle \Vdash_{\text{certify}} \mathsf{Id}$$

  and from the statement, $\Gamma \vdash_{(\mathrm{FO})} x(\overrightarrow{z} : \overrightarrow{\sigma^b}).\mathrm{P} : \mathsf{Proc}$. This deduction must have ended in zero or more uses of the weaken rule, preceded by an application of the input type rule with antecedent

  $$\Gamma''_{\overrightarrow{z}}, x : (\overrightarrow{\sigma^b})^c, \overrightarrow{z} : \overrightarrow{\sigma^b} \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}$$

  (where $\Gamma''_{\overrightarrow{z}}, x : (\overrightarrow{\sigma^b})^c$ is a subset of Γ). By the corollary to the substitution lemma (Corollary 4.1.2) then,

  $$\Gamma''_{\overrightarrow{z}}, x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma^b} \vdash_{\mathrm{FO}} \mathrm{P}\{\overrightarrow{y}/\overrightarrow{z}\} : \mathsf{Proc}$$

  Then by as many applications of the weaken rule as necessary,

  $$\Theta, \Gamma, \overrightarrow{y} : \overrightarrow{\sigma^b} \vdash_{(\mathrm{FO})} \mathrm{P} : \mathsf{Proc}$$

  as required (observing that the substitution generated is the identity substitution).

- Communication: The rule states

$$\frac{\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{(\nu \overrightarrow{z})\overline{x}[\overrightarrow{y}]} \langle \Theta', \Gamma', \mathrm{P}'\rangle \Vdash_{\text{certify}} \mathsf{Id} \quad \langle \Gamma, \Theta, \mathrm{Q}\rangle \xrightarrow{x(\overrightarrow{y})} \langle \Gamma', \Theta', \mathrm{Q}\rangle \Vdash_{\text{certify}} \mathbb{R} \qquad \overrightarrow{z} \notin FN(\mathrm{Q})}{\langle \Delta, \Gamma \cup \Theta, \mathrm{P}|\mathrm{Q}\rangle \xrightarrow{\tau} \langle \Delta, \Gamma' \cup \Theta, \mathrm{P}'|\mathrm{Q}'\rangle \Vdash_{\text{certify}} \mathbb{R}}$$

  From the statement, $\Gamma, \Theta \vdash_{\mathrm{FO}} \mathrm{P}|\mathrm{Q} : \mathsf{Proc}$. This deduction must have ended in zero or more uses of the weaken rule, preceded by an instance of the composition rule with antecedents

$$\begin{aligned} \Gamma'' &\vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc} \\ \text{and} \quad \Theta'' &\vdash_{\mathrm{FO}} \mathrm{Q} : \mathsf{Proc} \end{aligned}$$

  where Γ'' and Θ'' are subsets of Γ and Θ respectively. Then by as many applications of the weaken rule as necessary:

$$\begin{aligned} \Gamma &\vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc} \\ \text{and} \quad \Theta &\vdash_{\mathrm{FO}} \mathrm{Q} : \mathsf{Proc} \end{aligned}$$

  and so by the antecedents and the induction hypothesis (cases for output and input/certify)

$$\begin{aligned} \Gamma &\vdash_{\mathrm{FO}} \mathrm{P}' : \mathsf{Proc} \\ \text{and} \quad \mathbb{R}(\Theta &\vdash_{\mathrm{FO}} \mathrm{Q}' : \mathsf{Proc}) \end{aligned}$$

  (noting that the substitution in the first instance is the identity substitution). Hence by the composition rule

$$\mathbb{R}(\Gamma, \Theta \vdash_{\mathrm{FO}} \mathrm{P}'|\mathrm{Q}' : \mathsf{Proc})$$

  as required.

- Summation: The rule states

$$\frac{\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle \Vdash_{\text{certify}} \mathbb{R}}{\langle \Theta, \Gamma, \mathrm{P} + \mathrm{Q}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle \Vdash_{\text{certify}} \mathbb{R}}$$

  By the induction hypothesis the result holds for the antecedent; that is

$$\mathbb{R}(\Theta, \Gamma \vdash_{\mathrm{FO}} \mathrm{P}' : \mathsf{Proc})$$

  Hence the result holds directly for the summation case as well.

- Composition: The rule states

$$\frac{\langle \Theta, \Gamma, \mathrm{P} \rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}' \rangle \Vdash_{\text{certify}} \mathbb{R} \quad BN(\mu) \cap FN(\mathrm{Q}) = \emptyset}{\langle \Theta, \Gamma, \mathrm{P}|\mathrm{Q} \rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'|\mathrm{Q} \rangle \Vdash_{\text{certify}} \mathbb{R}}$$

  By the induction hypothesis the result holds for the antecedent; that is

$$\mathbb{R}(\Theta, \Gamma \vdash_{\mathrm{FO}} \mathrm{P}' : \mathsf{Proc})$$

  From the statement The rule states $\Gamma \vdash_{\mathrm{FO}} \mathrm{P}|\mathrm{Q} : \mathsf{Proc}$, the deduction of which much have ended in zero or more uses of the weaken rule, preceded by an application of the composition rule with antecedents $\Gamma_1 \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}$ and $\Gamma_2 \vdash_{\mathrm{FO}} \mathrm{Q} : \mathsf{Proc}$, where Γ_1 and Γ_2 are both subsets of Γ. Hence since Γ_2 is a subset of Γ, by the composition rule and the induction hypothesis result holds for $\Theta, \Gamma \vdash_{\mathrm{FO}} \mathrm{P}'|\mathrm{Q} : \mathsf{Proc}$ as required.

- Replication: similarly.

- Open: By the induction hypothesis and the reduction case and type rule for restriction.

- Certify as trusted: The rule states

$$\frac{\text{certify}(z) = \mathrm{T} \quad \mathbb{R} = \mathsf{Id}[i := \mathrm{T}]}{\begin{array}{l}\langle \Theta, \Gamma \cup \{x : (\sigma^i)^b\}, x(y)_{\text{certify}}^{?}\mathrm{P} \oplus \mathrm{Q} \rangle \xrightarrow{x(z)} \\ \qquad \langle \mathbb{R}(\Theta), \mathbb{R}(\Gamma \cup \{x : (\sigma^i)^b, z : \sigma^{\mathrm{T}}\}), \mathbb{R}(\mathrm{P}\{z/y\}) \rangle\end{array}}$$

  and from the statement $\Gamma \cup \{x : (\sigma^i)^b\} \vdash_{(\mathrm{FO})} x(y)_{\text{certify}}^{?}\mathrm{P} \oplus \mathrm{Q} : \mathsf{Proc}$. This deduction must have ended in zero or more uses of the weaken rule, preceded by an application of the certify rule with antecedents $\Gamma_1, x : (\sigma^i)^b, y : \sigma^{\mathrm{T}} \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}$ and $\Gamma_2, x : (\sigma^i)^b, y : \sigma^{\mathrm{U}} \vdash_{\mathrm{FO}} \mathrm{Q} : \mathsf{Proc}$ where Γ_1 and Γ_2 are subsets of Γ and $y \notin \text{dom}\Gamma$. Then by the substitution lemma (Lemma 4.1.1) $\Gamma_1, x : (\sigma^i)^b, z : \sigma^{\mathrm{T}} \vdash_{\mathrm{FO}} \mathrm{P}\{z/y\} : \mathsf{Proc}$, and hence if $\mathbb{R} = \mathsf{Id}[i := \mathrm{T}]$ then by as many applications of the weaken rule as necessary,

$$\mathbb{R}(\Theta, \Gamma, x : (\sigma^i)^b \vdash_{\mathrm{FO}} \mathrm{P}\{z/y\} : \mathsf{Proc})$$

  as required.

- Certify as untrusted: similarly.

□

The main result can now be stated as a refinement of this lemma to internal reductions only:

Theorem 4.1.4 (Subject Reduction) *If there is a valid deduction* $\Gamma \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}$ *in System* $\mathcal{T}_{\mathsf{FO}\pi}$*, and an internal reduction such that*

$$\langle \Delta, \Gamma, \mathrm{P} \rangle \xrightarrow{\tau} \langle \Delta, \Gamma', \mathrm{P}' \rangle \Vdash_{\mathrm{certify}} \mathbb{R}$$

then there is a valid deduction

$$\mathbb{R}(\Gamma) \vdash_{\mathrm{FO}} \mathbb{R}(\mathrm{P}') : \mathsf{Proc}$$

Proof. By Lemma 4.1.3, observing that the reduction must contain a use of the communication case (to construct an internal action, before any possible uses of the structural cases). □

A simple corollary of this states that types are preserved under multiple reduction steps (modulo, of course, the coercions involved in the reduction path).

Corollary 4.1.5 (Multi-Step Subject Reduction) *If there is a valid deduction* $\Gamma \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}$ *in System* $\mathcal{T}_{\mathsf{FO}\pi}$*, and an internal reduction path such that*

$$\langle \Delta, \Gamma, \mathrm{P} \rangle \xrightarrow{\overrightarrow{\tau}} \langle \Delta, \Gamma'', \mathrm{P}'' \rangle \Vdash_{\mathrm{certify}} \mathbb{R}$$

then there is a valid deduction

$$\mathbb{R}(\Gamma) \vdash_{\mathrm{FO}} \mathbb{R}(\mathrm{P}'') : \mathsf{Proc}$$

Proof. By induction on the reduction path, using the subject reduction Theorem (4.1.4).

Assume the reduction path is derived as follows:

$$\frac{\langle \Delta, \Gamma, \mathrm{P} \rangle \xrightarrow{\tau} \langle \Delta, \Gamma', \mathrm{P}' \rangle \Vdash_{\mathrm{certify}} \mathbb{R} \quad \langle \Delta, \Gamma', \mathrm{P}' \rangle \xrightarrow{\overrightarrow{\tau}} \langle \Delta, \Gamma'', \mathrm{P}'' \rangle \Vdash_{\mathrm{certify}} \mathbb{R}'}{\langle \Delta, \Gamma, \mathrm{P} \rangle \xrightarrow{\tau \overrightarrow{\tau}} \langle \Delta, \Gamma'', \mathrm{P}'' \rangle \Vdash_{\mathrm{certify}} \mathbb{R}; \mathbb{R}'}$$

Then by Theorem 4.1.4 (subject reduction) $\mathbb{R}(\Gamma) \vdash_{\mathrm{FO}} \mathbb{R}(\mathrm{P}') : \mathsf{Proc}$, and by the induction hypothesis the corollary holds for $\langle \Delta, \Gamma', \mathrm{P}' \rangle \xrightarrow{\overrightarrow{\tau}} \langle \Delta, \Gamma'', \mathrm{P}'' \rangle \Vdash_{\mathrm{certify}} \mathbb{R}'$ as well; that is $\mathbb{R}'(\mathbb{R}(\Gamma)) \vdash_{\mathrm{FO}} \mathbb{R}'(\mathbb{R}(\mathrm{P}'')) : \mathsf{Proc}$. Hence corollary holds for $\langle \Delta, \Gamma, \mathrm{P} \rangle \xrightarrow{\tau \overrightarrow{\tau}} \langle \Delta, \Gamma'', \mathrm{P}'' \rangle \Vdash_{\mathrm{certify}} \mathbb{R}; \mathbb{R}'$ (recalling that $\mathbb{R}'(\mathbb{R}(\Gamma)) = \mathbb{R}; \mathbb{R}'(\Gamma)$ as required). □

4.2 Safety of System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$

4.2.1 Substitution Lemma

The Substitution Lemma for System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$ is more or less unchanged from the base statement for System $\mathcal{T}_{\mathsf{FO}\pi}$ in Section 4.1.1 with the only major difference being a slight relaxation, in the form of a subtype constraint, on the type of the substitute name. This causes very little difficulty in the proofs; the construction of guarded processes are the only times at which extra care must be taken, and in general the observation that transitivity of $\leq_{\mathsf{B}}$ ensures the security constraint is satisfied by a subtype is sufficient.

Lemma 4.2.1 (Substitution Lemma) *If there is a valid deduction*

$$\Gamma_{xy}, x : \sigma^b \vdash^c_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}$$

in System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$, then there is also a valid deduction for

$$\Gamma_{xy}, y : \sigma^{b'} \vdash^c_{\mathrm{FO}'\leq} \mathrm{P}\,\{y/x\} : \mathsf{Proc}$$

where $\sigma^{b'} \leq \sigma^b$.

Proof. By induction on the derivation of $\Gamma_{xy}, x : \sigma^b \vdash^c_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}$.

- Zero: The axiom as it appears in the rules

  $$\frac{}{\emptyset \vdash^c_{(\mathrm{FO}'\leq)} \mathbf{0} : \mathsf{Proc}}$$

  is not in the form required (with the domain of the environment containing x), however if the environment is extended using the weaken rule it may be noted that (by Definition 2.2.6) $\mathbf{0}\{y/x\} = \mathbf{0}$ for every x and y as required.

- Weaken: The rule has the form

  $$\frac{\Gamma \vdash^c_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc} \quad z \notin \mathrm{dom}\Gamma \quad \vdash \sigma'^d}{\Gamma, z : \sigma'^d \vdash^c_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}}$$

  Assuming $\Gamma = \Gamma'_{xy}, x : \sigma^b$ for some Γ', then by the induction hypothesis $\Gamma'_{xy}, y : \sigma^b \vdash^c_{\mathrm{FO}'\leq} \mathrm{P}\{y/x\} : \mathsf{Proc}$ is valid. Hence by the weaken rule $\Gamma'_{xy}, y : \sigma^{b'}, z : \sigma'^d \vdash^c_{\mathrm{FO}'\leq} \mathrm{P}\{y/x\} : \mathsf{Proc}$ is also valid given $\sigma^{b'} \leq \sigma^b$, noting that $z \neq x, y$ by the clause in the weaken rule.

- Replication: By the induction hypothesis on the antecedent and the replication rule.

- Summation: The rule has the form

$$\frac{\Gamma \vdash^{c}_{(\mathrm{FO}'\leq)} \mathrm{P} : \mathsf{Proc} \quad \Theta \vdash^{c}_{(\mathrm{FO}'\leq)} \mathrm{Q} : \mathsf{Proc}}{\Gamma, \Theta \vdash^{c}_{(\mathrm{FO}'\leq)} \mathrm{P} + \mathrm{Q} : \mathsf{Proc}}$$

  By the induction hypothesis the result holds for both the antecedents individually, that is

$$\Gamma'_{xy}, y : \sigma^{b'} \vdash^{c}_{\mathrm{FO}'\leq} \mathrm{P}\{y/x\} : \mathsf{Proc} \quad \text{and} \quad \Theta'_{xy}, y : \sigma^{b'} \vdash^{c}_{\mathrm{FO}'\leq} \mathrm{Q}\{y/x\} : \mathsf{Proc}$$

  where $\Gamma = \Gamma'_{xy}, x : \sigma^{b}$ and $\Theta = \Theta'_{xy}, x : \sigma^{b}$ for some Γ' and Θ', and $\sigma^{b'} \leq \sigma^{b}$. Hence by the summation rule and the definition of substitution,

$$\Gamma'_{xy}, \Theta'_{xy}, y : \sigma^{b'} \vdash^{c}_{\mathrm{FO}'\leq} (\mathrm{P} + \mathrm{Q})\,\{y/x\} : \mathsf{Proc}$$

  as required.

- Composition: Similarly.

- Restriction: By the induction hypothesis, noting that due to the assumption of Barendregt's Convention (Page 21) and the statement which has y free in the consequent the case where x or y is the restricted name need not be considered.

- Conversion: Trivial, by the induction hypothesis.

- Output: The rule has the form

$$\frac{\Gamma, w : (\overrightarrow{\sigma^{c}})^{d}, \overrightarrow{z} : \overrightarrow{\sigma'^{c'}} \vdash^{e}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc} \quad \overrightarrow{\sigma'^{c'}} \leq \overrightarrow{\sigma^{c}} \quad e \leq_{\mathsf{B}} d}{\Gamma, w : (\overrightarrow{\sigma^{c}})^{d}, \overrightarrow{z} : \overrightarrow{\sigma'^{c'}} \vdash^{e}_{(\mathrm{FO}'\leq)} \overline{w}[\overrightarrow{z}].\mathrm{P} : \mathsf{Proc}}$$

  By the induction hypothesis the result holds for the antecedent, in other words assuming $\Gamma = \Gamma'_{xy}, x : \sigma^{b}$ then $\Gamma'_{xy}, y : \sigma^{b'}, w : (\overrightarrow{\sigma'^{c}})^{d}, \overrightarrow{z} : \overrightarrow{\sigma'^{c'}} \vdash^{e}_{\mathrm{FO}'\leq} \mathrm{P}\{y/x\} : \mathsf{Proc}$ is valid. There are three cases to consider in the consequent; x and y distinct from w and $\overrightarrow{z}$, $x = w$, and $x \in \overrightarrow{z}$ (note that due to the lack of recursive types the case where both $x = w$ *and* $x \in \overrightarrow{z}$ cannot occur). The first holds easily by the output rule, observing that by definition $\sigma^{b'} \leq \sigma^{b}$ implies $b \leq_{\mathsf{B}} b'$, and hence by transitivity of $\leq_{\mathsf{B}}$ the requirement that $e \leq_{\mathsf{B}} b'$ is maintained. The second case follows the use of induction hypothesis to give

$\Gamma'_{xy}, y : (\overrightarrow{\sigma'^{c'}})^d, \overrightarrow{z} : \overrightarrow{\sigma'^{c}} \vdash^{e}_{\mathrm{FO}'\leq} \overline{y}[\overrightarrow{z}].\mathrm{P}\{y/x\} : \mathsf{Proc}$, recalling from Definition 2.2.6 that $(\overline{x}[\overrightarrow{z}].\mathrm{P})\{y/x\} = \overline{y}[\overrightarrow{z}].(\mathrm{P})\{y/x\})\{y/x\}$ when $x \notin \overrightarrow{z}$. The third case follows similarly, observing that transitivity of $\leq$ means the subtype condition is preserved.

- Input: Similarly (noting that by the statement x or y cannot appear in the bound input names).
- Certify: Similarly, noting that the bound name cannot be x or y.

□

Corollary 4.2.2 (Substitution Corollary) *If there is a valid deduction*

$$\Gamma_{\overrightarrow{xy}}, \overrightarrow{x} : \overrightarrow{\sigma^b} \vdash^{c}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}$$

in System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$, where all $\overrightarrow{x}$ are distinct, then given $\overrightarrow{\sigma^{b'}} \leq \overrightarrow{\sigma^b}$

$$\Gamma_{\overrightarrow{xy}}, \overrightarrow{y} : \overrightarrow{\sigma^{b'}} \vdash^{c}_{\mathrm{FO}'\leq} \mathrm{P}\,\{\overrightarrow{y}/\overrightarrow{x}\} : \mathsf{Proc}$$

is also a valid deduction.

Proof. By Lemma 4.2.1 (the Substitution Lemma). □

4.2.2 Subject Reduction

Like the substitution lemma, the statement of the subject reduction theorem is similar to the corresponding result for System $\mathcal{T}_{\mathsf{FO}\pi}$, with the added need to consider sub-typing and the security level.

The presentation is also similar; the majority of the proof is contained in a lemma, with the main result using this lemma.

Lemma 4.2.3 *If there is a valid deduction $\Gamma \vdash^{b}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}$ in System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$, and a reduction such that*

$$\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle \Vdash_{\mathrm{certify}} \mathbb{R}$$

then there is a valid deduction

$$\mathbb{R}(\Gamma \cup \Theta) \vdash^{\mathbb{R}(b)}_{\mathrm{FO}'\leq} \mathbb{R}(\mathrm{P}') : \mathsf{Proc}$$

Note that both environments are used in the second reduction, to incorporate any names acquired (such as via an input) from the anonymous process. Similarly for $\mathbb{R}(\Gamma \cup \Theta) \vdash^{\mathbb{R}(b)}_{(\mathrm{FO}'\leq)} \mathbb{R}(\mathrm{P}') : \mathsf{Proc}$

Proof. By induction on the derivation of $\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle \Vdash_{\text{certify}} \mathbb{R}$ and on the deduction $\Gamma \vdash^{b}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}$.

- Output: The rule states

  $$\langle \Theta, \Gamma \cup \{\overrightarrow{y} : \overrightarrow{\sigma^b}\}, \overline{x}[\overrightarrow{y}].\mathrm{P}\rangle \xrightarrow{\overline{x}[\overrightarrow{y}]} \langle \Theta \cup \{\overrightarrow{y} : \overrightarrow{\sigma^b}\}, \Gamma \cup \{\overrightarrow{y} : \overrightarrow{\sigma^b}\}, \mathrm{P}\rangle \Vdash_{\text{certify}} \mathsf{Id}$$

  and from the statement $\Gamma, \overrightarrow{y} : \overrightarrow{\sigma^b} \vdash^{b}_{(\mathrm{FO}'\leq)} \overline{x}[\overrightarrow{y}].\mathrm{P} : \mathsf{Proc}$. This deduction must have ended in zero or more uses of the weaken rule, preceded by an application of the output type rule with antecedent

  $$\Gamma'', x : (\overrightarrow{\sigma^{b'}})^c, \overrightarrow{y} : \overrightarrow{\sigma^b} \vdash^{d}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}$$

  (where $\Gamma'', x : (\overrightarrow{\sigma^b})^c$ is a subset of Γ, $d \leq_{\mathsf{B}} c$, and $\overrightarrow{\sigma^b} \leq \overrightarrow{\sigma^{b'}}$). Then by as many applications of the weaken rule as necessary,

  $$\Theta, \Gamma, \overrightarrow{y} : \overrightarrow{\sigma^b} \vdash^{d}_{(\mathrm{FO}'\leq)} \mathrm{P} : \mathsf{Proc}$$

  as required (observing that the substitution generated is the identity substitution).

- Input: The rule states

  $$\langle \Theta \cup \{\overrightarrow{y} : \overrightarrow{\sigma^b}\}, \Gamma, x(\overrightarrow{z} : \overrightarrow{\sigma^b}).\mathrm{P}\rangle \xrightarrow{x(\overrightarrow{y})} \langle \Theta \cup \{\overrightarrow{y} : \overrightarrow{\sigma^b}\}, \Gamma \cup \{\overrightarrow{y} : \overrightarrow{\sigma^b}\}, \mathrm{P}\{\overrightarrow{y}/\overrightarrow{z}\}\rangle \Vdash_{\text{certify}} \mathsf{Id}$$

  and from the statement, $\Gamma \vdash^{d}_{(\mathrm{FO}'\leq)} x(\overrightarrow{z} : \overrightarrow{\sigma^b}).\mathrm{P} : \mathsf{Proc}$. This deduction must have ended in zero or more uses of the weaken rule, preceded by an application of the input type rule with antecedent

  $$\Gamma''_{\overrightarrow{z}}, x : (\overrightarrow{\sigma^b})^c, \overrightarrow{z} : \overrightarrow{\sigma^{b'}} \vdash^{d}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}$$

  (where $\Gamma''_{\overrightarrow{z}}, x : (\overrightarrow{\sigma^b})^c$ is a subset of Γ, $\overrightarrow{\sigma^b} \leq \overrightarrow{\sigma^{b'}}$, and $d \leq_{\mathsf{B}} c$). By the corollary to the substitution lemma (Corollary 4.2.2) then, given $\overrightarrow{\sigma^{b''}} \leq \overrightarrow{\sigma^b}$,

  $$\Gamma''_{\overrightarrow{z}}, x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma^{b''}} \vdash^{b}_{\mathrm{FO}'\leq} \mathrm{P}\{\overrightarrow{y}/\overrightarrow{z}\} : \mathsf{Proc}$$

  Then by as many applications of the weaken rule as necessary,

  $$\Theta, \Gamma, \overrightarrow{y} : \overrightarrow{\sigma^b} \vdash^{b}_{(\mathrm{FO}'\leq)} \mathrm{P}\{\overrightarrow{y}/\overrightarrow{z}\} : \mathsf{Proc}$$

  as required (observing that the substitution generated is the identity substitution).

- Communication: The rule states

$$\frac{\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{(\nu \overrightarrow{z})\overline{x}[\overrightarrow{y}]} \langle \Theta', \Gamma', \mathrm{P}'\rangle \Vdash_{\text{certify}} \mathsf{Id} \quad \langle \Gamma, \Theta, \mathrm{Q}\rangle \xrightarrow{x(\overrightarrow{y})} \langle \Gamma', \Theta', \mathrm{Q}\rangle \Vdash_{\text{certify}} \mathbb{R} \qquad \overrightarrow{z} \notin FN(\mathrm{Q})}{\langle \Delta, \Gamma \cup \Theta, \mathrm{P}|\mathrm{Q}\rangle \xrightarrow{\tau} \langle \Delta, \Gamma' \cup \Theta, \mathrm{P}'|\mathrm{Q}'\rangle \Vdash_{\text{certify}} \mathbb{R}}$$

  From the statement, $\Gamma, \Theta \vdash^{b}_{\mathrm{FO}'\leq} \mathrm{P}|\mathrm{Q} : \mathsf{Proc}$. This deduction must have ended in zero or more uses of the weaken and sub-typing rules, preceded by an instance of the composition rule with antecedents

$$\begin{aligned} \Gamma'' \quad &\vdash^{c'}_{\mathrm{FO}'\leq} \quad \mathrm{P} : \mathsf{Proc} \\ \text{and} \quad \Theta'' \quad &\vdash^{d'}_{\mathrm{FO}'\leq} \quad \mathrm{Q} : \mathsf{Proc} \end{aligned}$$

  where Γ'' and Θ'' are subsets of Γ and Θ respectively, and $c' \cdot d' = b' \geq_{\mathsf{B}} b$. Then by as many applications of the weaken and sub-typing rules as necessary:

$$\begin{aligned} \Gamma \quad &\vdash^{c}_{\mathrm{FO}'\leq} \quad \mathrm{P} : \mathsf{Proc} \\ \text{and} \quad \Theta \quad &\vdash^{d}_{\mathrm{FO}'\leq} \quad \mathrm{Q} : \mathsf{Proc} \end{aligned}$$

  and so by the antecedents and the induction hypothesis (cases for output and input/certify)

$$\begin{aligned} \Gamma \quad &\vdash^{c}_{\mathrm{FO}'\leq} \quad \mathrm{P}' : \mathsf{Proc} \\ \text{and} \quad \mathbb{R}(\Theta \quad &\vdash^{d}_{\mathrm{FO}'\leq} \quad \mathrm{Q}' : \mathsf{Proc}) \end{aligned}$$

  (noting that the substitution in the first instance is the identity substitution). Hence by the composition rule

$$\mathbb{R}(\Gamma, \Theta \vdash^{b}_{\mathrm{FO}'\leq} \mathrm{P}'|\mathrm{Q}' : \mathsf{Proc})$$

  where $b = c \cdot d$ as required.

- Summation: The rule states

$$\frac{\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle \Vdash_{\text{certify}} \mathbb{R}}{\langle \Theta, \Gamma, \mathrm{P} + \mathrm{Q}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle \Vdash_{\text{certify}} \mathbb{R}}$$

  By the induction hypothesis the result holds for the antecedent; that is

$$\mathbb{R}(\Theta, \Gamma \vdash^{b}_{\mathrm{FO}'\leq} \mathrm{P}' : \mathsf{Proc})$$

  Hence the result holds directly for the summation case as well.

- Composition: The rule states

$$\frac{\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle \Vdash_{\mathrm{certify}} \mathbb{R} \quad BN(\mu) \cap FN(\mathrm{Q}) = \emptyset}{\langle \Theta, \Gamma, \mathrm{P}|\mathrm{Q}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'|\mathrm{Q}\rangle \Vdash_{\mathrm{certify}} \mathbb{R}}$$

  By the induction hypothesis the result holds for the antecedent; that is

$$\mathbb{R}(\Theta, \Gamma \vdash^{b}_{\mathrm{FO}'\leq} \mathrm{P}' : \mathsf{Proc})$$

  From the statement $\Gamma \vdash^{bc}_{\mathrm{FO}'\leq} \mathrm{P}|\mathrm{Q} : \mathsf{Proc}$, the deduction of which much have ended in zero or more uses of the weaken rule, preceded by an application of the composition rule with antecedents $\Gamma_1 \vdash^{b}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}$ and $\Gamma_2 \vdash^{c}_{\mathrm{FO}'\leq} \mathrm{Q} : \mathsf{Proc}$, where Γ_1 and Γ_2 are both subsets of Γ. Hence since Γ_2 is a subset of Γ, by the composition rule and the induction hypothesis result holds for $\Theta, \Gamma \vdash^{bc}_{\mathrm{FO}'\leq} \mathrm{P}'|\mathrm{Q} : \mathsf{Proc}$ as required.

- Replication: Similarly.

- Open: By the induction hypothesis, the case for output, and the type rule for restriction.

- Certify as trusted: The rule states

$$\frac{\mathrm{certify}(z) = \mathrm{T} \quad \mathbb{R} = \mathsf{Id}[i := \mathrm{T}]}{\begin{array}{l}\langle \Theta, \Gamma \cup \{x : (\sigma^i)^b\}, x(y)_{\mathrm{certify}}?\mathrm{P} \oplus \mathrm{Q}\rangle \xrightarrow{x(z)} \\ \qquad \langle \mathbb{R}(\Theta), \mathbb{R}(\Gamma \cup \{x : (\sigma^i)^b, z : \sigma^{\mathrm{T}}\}), \mathbb{R}(\mathrm{P}\{z/y\})\rangle\end{array}}$$

  and from the statement $\Gamma \cup \{x : (\sigma^i)^b\} \vdash^{ic+\bar{i}d}_{(\mathrm{FO}'\leq)} x(y)_{\mathrm{certify}}?\mathrm{P} \oplus \mathrm{Q} : \mathsf{Proc}$. This deduction must have ended in zero or more uses of the weaken rule, followed by an application of the certify rule with antecedents $\Gamma_1, x : (\sigma^i)^b, y : \sigma^{\mathrm{T}} \vdash^{c}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}$ and $\Gamma_2, x : (\sigma^i)^b, y : \sigma^{\mathrm{U}} \vdash^{d}_{\mathrm{FO}'\leq} \mathrm{Q} : \mathsf{Proc}$ where Γ_1 and Γ_2 are subsets of Γ, $y \notin \mathrm{dom}\Gamma$, and $c \leq_{\mathsf{B}} b$, $d \leq_{\mathsf{B}} b$. Then by the substitution lemma (Lemma 4.2.1) $\Gamma_1, x : (\sigma^i)^b, z : \sigma^{\mathrm{T}} \vdash^{b}_{\mathrm{FO}'\leq} \mathrm{P}\{z/y\} : \mathsf{Proc}$, and hence if $\mathbb{R} = \mathsf{Id}[i := \mathrm{T}]$ then by as many applications of the weaken rule as necessary,

$$\mathbb{R}(\Theta, \Gamma, x : (\sigma^i)^b \vdash^{c}_{\mathrm{FO}'\leq} \mathrm{P}\{z/y\} : \mathsf{Proc})$$

  as required.

- Certify as untrusted: Similarly.

□

The main result can now be stated as a refinement of this lemma to internal reductions only:

Theorem 4.2.4 (Subject Reduction) *If there is a valid deduction*

$$\Gamma \vdash^{b}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}$$

in System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$*, and an internal reduction such that*

$$\langle \Delta, \Gamma, \mathrm{P}\rangle \xrightarrow{\tau} \langle \Delta, \Gamma', \mathrm{P}'\rangle \Vdash_{\mathrm{certify}} \mathbb{R}$$

then there is a valid deduction

$$\mathbb{R}(\Gamma) \vdash^{\mathbb{R}(b)}_{\mathrm{FO}'\leq} \mathbb{R}(\mathrm{P}') : \mathsf{Proc}$$

Proof. By Lemma 4.2.3, observing that the reduction must contain a use of the communication case (to construct an internal action, before any possible uses of the structural cases). □

A simple corollary of this states that types are preserved under multiple reduction steps (modulo, of course, the coercions involved in the reduction path).

Corollary 4.2.5 (Multi-Step Subject Reduction) *If there is a valid deduction* $\Gamma \vdash^{b}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}$ *in System* $\mathcal{T}_{\mathsf{FO}'\pi\leq}$*, and an internal reduction path such that*

$$\langle \Delta, \Gamma, \mathrm{P}\rangle \xrightarrow{\overrightarrow{\tau}} \langle \Delta, \Gamma'', \mathrm{P}''\rangle \Vdash_{\mathrm{certify}} \mathbb{R}$$

then there is a valid deduction

$$\mathbb{R}(\Gamma) \vdash^{b}_{\mathrm{FO}'\leq} \mathbb{R}(\mathrm{P}'') : \mathsf{Proc}$$

Proof. By induction on the reduction path, using the subject reduction Theorem (4.2.4).

Assume the reduction path is derived as follows:

$$\frac{\langle \Delta, \Gamma, \mathrm{P}\rangle \xrightarrow{\tau} \langle \Delta, \Gamma', \mathrm{P}'\rangle \Vdash_{\mathrm{certify}} \mathbb{R} \quad \langle \Delta, \Gamma', \mathrm{P}'\rangle \xrightarrow{\overrightarrow{\tau}} \langle \Delta, \Gamma'', \mathrm{P}''\rangle \Vdash_{\mathrm{certify}} \mathbb{R}'}{\langle \Delta, \Gamma, \mathrm{P}\rangle \xrightarrow{\tau\overrightarrow{\tau}} \langle \Delta, \Gamma'', \mathrm{P}''\rangle \Vdash_{\mathrm{certify}} \mathbb{R};\mathbb{R}'}$$

Then by Theorem 4.2.4 (subject reduction) $\mathbb{R}(\Gamma) \vdash^{b}_{\mathrm{FO}'\leq} \mathbb{R}(\mathrm{P}') : \mathsf{Proc}$, and by the induction hypothesis the corollary holds for $\langle \Delta, \Gamma', \mathrm{P}'\rangle \xrightarrow{\overrightarrow{\tau}} \langle \Delta, \Gamma'', \mathrm{P}''\rangle \Vdash_{\mathrm{certify}} \mathbb{R}'$ as well; that is $\mathbb{R}'(\mathbb{R}(\Gamma)) \vdash^{b}_{\mathrm{FO}'\leq} \mathbb{R}'(\mathbb{R}(\mathrm{P}'')) : \mathsf{Proc}$. Hence corollary holds for $\langle \Delta, \Gamma, \mathrm{P}\rangle \xrightarrow{\tau\overrightarrow{\tau}} \langle \Delta, \Gamma'', \mathrm{P}''\rangle \Vdash_{\mathrm{certify}} \mathbb{R};\mathbb{R}'$ (recalling that $\mathbb{R}'(\mathbb{R}(\Gamma)) = \mathbb{R};\mathbb{R}'(\Gamma)$ as required). □

4.2.3 Security Properties

The security property presented here is a form of non-interference; that low-integrity channels cannot disrupt (for example, by blocking) a process typed at a higher integrity level. The precise formation of the property is Strong Non-deterministic Non-Interference or SNNI (Focardi and Gorrieri 1995). Essentially, it states that for every trace derivable for a process typed at a certain level, the trace obtained by removing all labels with a lower integrity is also a valid trace.

For example, consider the process $x.x.\mathbf{0}|y.\mathbf{0}$, where x is trusted and y is untrusted. Traces possible for this process include xyx and yxx. Note that for each of these the trace obtained by removing the untrusted label y, giving xx in both cases, is also a valid trace. Conversely the process $x.y.x.\mathbf{0}$ does *not* satisfy this property, as the trace xx (derived by deleting y from xyx again) is not valid.

There are three steps in the proof of this property for System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$: the proof of a stronger single-step property that *no* labels with a trustedness lower than the security level of the process in question can exist in any trace, followed by the corresponding multi-step proof, and lastly the proof of SNNI which is a simple consequence of the stronger result.

Firstly, the single-step result. Note the formation of the final clause in the statement, which combines both environments and incorporates the coercion substitution. An alternative presentation such as $\Gamma'(\|\mu\|)$ might be just as accurate, but more cumbersome to prove. It is also unnecessary in the single step result to combine the environments as the channel used must be present in Γ for the action to take place, but it is presented this way for consistency with the multi-step result which does require the combination.

Theorem 4.2.6 *For any valid deduction* $\Gamma \vdash^{b}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}$ *in System* $\mathcal{T}_{\mathsf{FO}'\pi\leq}$*, for each action* $\mu \neq \tau$ *such that* $\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle \Vdash_{\mathrm{certify}} \mathbb{R}$ *there is some* σ^c *with* $c \geq_{\mathsf{B}} b$ *such that* $\mathbb{R}(\Gamma \cup \Theta)(\|\mu\|) = \sigma^c$*.*

Proof. By induction on the derivation of $\langle \Theta, \Gamma, \mathrm{P}\rangle \xrightarrow{\mu} \langle \Theta', \Gamma', \mathrm{P}'\rangle \Vdash_{\mathrm{certify}} \mathbb{R}$, and by cases on the derivation of $\Gamma \vdash^{b}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}$.

There are four main cases that need to be considered; those for input, output, and coercion as trusted and untrusted. The remaining structural induction rules of alpha-congruence, summation, composition, restriction, replication, and open largely follow by the induction hypothesis. The rule for communication need not be considered as it generates an anonymous action of τ.

- Input: by definition,

$$\langle \Theta \cup \{\overrightarrow{z} : \overrightarrow{\sigma^d}\}, \Gamma, x(\overrightarrow{y} : \overrightarrow{\sigma^d}).\mathrm{P}\rangle \xrightarrow{x(\overrightarrow{z})} \langle \Theta \cup \{\overrightarrow{z} : \overrightarrow{\sigma^d}\}, \Gamma \cup \{\overrightarrow{z} : \overrightarrow{\sigma^d}\}, \mathrm{P}\{\overrightarrow{z}/\overrightarrow{y}\}\rangle \Vdash_{\text{certify}} \mathsf{Id}$$

  From the statement,

$$\Gamma \vdash^{b}_{\mathrm{FO}'\leq} x(\overrightarrow{y}).\mathrm{P} : \mathsf{Proc}$$

  This deduction must have ended in a use of the input rule, followed by zero or more uses of the relaxation and weaken rules. Since weaken doesn't affect the types, and transitivity of $\leq_{\mathsf{B}}$ ensures that the relaxation rule doesn't affect the requirements on the guard annotations, the most pertinent is the input rule, which must have had the antecedent

$$\Gamma'_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^d})^c, \overrightarrow{y} : \overrightarrow{\sigma'^{d'}} \vdash^{b'}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}$$

  where $\overrightarrow{\sigma^d} \leq \overrightarrow{\sigma'^{d'}}$, and importantly $b' \leq_{\mathsf{B}} c$. Hence $(\Gamma \cup \Theta)(x) = (\overrightarrow{\sigma^d})^c$, and since any use of the relaxation rule means $b \leq_{\mathsf{B}} b'$, transitivity implies $b \leq_{\mathsf{B}} c$ as required.

- Output: by definition,

$$\langle \Theta, \Gamma \cup \{\overrightarrow{y} : \overrightarrow{\sigma^d}\}, \overline{x}[\overrightarrow{y}].\mathrm{P}\rangle \xrightarrow{\overline{x}[\overrightarrow{y}]} \langle \Theta \cup \{\overrightarrow{y} : \overrightarrow{\sigma^d}\}, \Gamma \cup \{\overrightarrow{y} : \overrightarrow{\sigma^d}\}, \mathrm{P}\rangle \Vdash_{\text{certify}} \mathsf{Id}$$

  From the statement,

$$\Gamma \vdash^{b}_{\mathrm{FO}'\leq} \overline{x}[\overrightarrow{y}].\mathrm{P} : \mathsf{Proc}$$

  This deduction must have ended in a use of the output rule, followed by zero or more uses of the relaxation and weaken rules. Since weaken doesn't affect the types, and transitivity of $\leq_{\mathsf{B}}$ ensures that the relaxation rule doesn't affect the requirements on the guard annotations, the most pertinent is the output rule, which must have had the antecedent

$$\Gamma, x : (\overrightarrow{\sigma^d})^c, \overrightarrow{y} : \overrightarrow{\sigma'^{d'}} \vdash^{b'}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}$$

  where $\overrightarrow{\sigma'^{b'}} \leq \overrightarrow{\sigma^b}$, and importantly $b' \leq_{\mathsf{B}} c$. Hence $(\Gamma \cup \Theta)(x) = (\overrightarrow{\sigma^d})^c$, and since any use of the relaxation rule means $b \leq_{\mathsf{B}} b'$, transitivity implies $b \leq_{\mathsf{B}} c$ as required.

- Certify as trusted: by definition,

$$\frac{\Gamma(x) = (\sigma^i)^b \quad \text{certify}(z) = \mathrm{T} \quad \mathbb{R} = \mathsf{Id}[i := \mathrm{T}]}{\langle \Theta, \Gamma, x(y)_{\text{certify}}?\mathrm{P} \oplus \mathrm{Q} \rangle \xrightarrow{x(z)} \langle \mathbb{R}(\Theta), \mathbb{R}(\Gamma \cup \{z : \sigma^{\mathrm{T}}\}), \mathbb{R}(\mathrm{P}\{z/y\}) \rangle}$$

  From the statement,

$$\Gamma \vdash^{b}_{\mathrm{FO}'\leq} x(y)_{\text{certify}}?\mathrm{P} \oplus \mathrm{Q} : \mathsf{Proc}$$

  This deduction must have ended in a use of the certify rule, followed by zero or more uses of the relaxation and weaken rules. Since weaken doesn't affect the types, and transitivity of $\leq_{\mathsf{B}}$ ensures that the relaxation rule doesn't affect the requirements on the guard annotations, the most pertinent is the certify rule, which has the form

$$\frac{\begin{array}{ll}\Gamma_{1y}, x : (\sigma^i)^c, y : \sigma^{\mathrm{T}} \vdash^{d}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc} & d \leq_{\mathsf{B}} c \\ \Gamma_{2y}, x : (\sigma^i)^c, y : \sigma^{\mathrm{U}} \vdash^{e}_{\mathrm{FO}'\leq} \mathrm{Q} : \mathsf{Proc} & e \leq_{\mathsf{B}} c\end{array}}{\Gamma_{1y}, \Gamma_{2y}, x : (\sigma^i)^c \vdash^{id+\bar{i}e}_{(\mathrm{FO}'\leq)} x(y : \sigma^i)_{\text{certify}}?\mathrm{P} \oplus \mathrm{Q} : \mathsf{Proc}}$$

  where $b' = id + \bar{i}e$ and $\Gamma = \Gamma_{1y}, \Gamma_{2y}, x : (\sigma^i)^c$. Then $\mathbb{R}(\Gamma \cup \Theta)(x) = \mathbb{R}((\sigma^i)^c)$, so applying the substitution through, $\mathbb{R}(b) = \mathbb{R}(id + \bar{i}e) = \mathbb{R}(d)$ and by the type rule above, $\mathbb{R}(d) \leq_{\mathsf{B}} \mathbb{R}(c)$ as required.

- Certify as untrusted: by definition,

$$\frac{\Gamma(x) = (\sigma^i)^b \quad \text{certify}(z) = \mathrm{U} \quad \mathbb{R} = \mathsf{Id}[i := \mathrm{U}]}{\langle \Theta, \Gamma, x(y)_{\text{certify}}?\mathrm{P} \oplus \mathrm{Q} \rangle \xrightarrow{x(z)} \langle \mathbb{R}(\Theta), \mathbb{R}(\Gamma \cup \{z : \sigma^{\mathrm{T}}\}), \mathbb{R}(\mathrm{Q}\{z/y\}) \rangle}$$

  From the statement,

$$\Gamma \vdash^{b}_{\mathrm{FO}'\leq} x(y)_{\text{certify}}?\mathrm{P} \oplus \mathrm{Q} : \mathsf{Proc}$$

  This deduction must have ended in a use of the certify rule, followed by zero or more uses of the relaxation and weaken rules. Since weaken doesn't affect the types, and transitivity of $\leq_{\mathsf{B}}$ ensures that the relaxation rule doesn't affect the requirements on the guard annotations, the most pertinent is the certify rule, which has the form

$$\frac{\begin{array}{ll}\Gamma_{1y}, x : (\sigma^i)^c, y : \sigma^{\mathrm{T}} \vdash^{d}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc} & d \leq_{\mathsf{B}} c \\ \Gamma_{2y}, x : (\sigma^i)^c, y : \sigma^{\mathrm{U}} \vdash^{e}_{\mathrm{FO}'\leq} \mathrm{Q} : \mathsf{Proc} & e \leq_{\mathsf{B}} c\end{array}}{\Gamma_{1y}, \Gamma_{2y}, x : (\sigma^i)^c \vdash^{id+\bar{i}e}_{(\mathrm{FO}'\leq)} x(y : \sigma^i)_{\text{certify}}?\mathrm{P} \oplus \mathrm{Q} : \mathsf{Proc}}$$

where $b' = id + \bar{i}e$ and $\Gamma = \Gamma_{1y}, \Gamma_{2y}, x : (\sigma^i)^c$. Then $\mathbb{R}(\Gamma \cup \Theta)(x) = \mathbb{R}((\sigma^i)^c)$, so applying the substitution through, $\mathbb{R}(b) = \mathbb{R}(id + \bar{i}e) = \mathbb{R}(e)$ and by the type rule above, $\mathbb{R}(e) \leq_{\mathsf{B}} \mathbb{R}(c)$ as required.

The cases for alpha congruence, summation, replication, restriction, and open follow directly from the induction hypothesis. The case for composition follows similarly, noting that the meet (multiplication) condition on the security levels means the relevant level (of the deduction of the process directly involved in the reduction) may only be relaxed, and hence transitivity of $\leq_{\mathsf{B}}$ again means the result holds. □

The multi-step extension of this result holds as a trivial extension:

Corollary 4.2.7 *For any valid deduction* $\Gamma \vdash^{b}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}$ *in System* $\mathcal{T}_{\mathsf{FO}'\pi\leq}$, *then for each trace* $\langle \Theta, \Gamma, \mathrm{P} \rangle \xrightarrow{\vec{\mu}} \langle \Theta', \Gamma', \mathrm{P}' \rangle \Vdash_{\mathrm{certify}} \mathbb{R}$ *there is some* σ^c *with* $c \geq_{\mathsf{B}} b$ *such that* $\mathbb{R}(\Gamma \cup \Theta)(\|\mu\|) = \sigma^c$ *for each label* $\mu \in \vec{\mu}$, $\mu \neq \tau$.

Proof. By Theorem 4.2.6 and the induction hypothesis. □

Finally, the result for Strong Non-deterministic Non-Interference can be stated. The proof is trivial, as the property is a subset of Corollary 4.2.7. First, define a refinement of the ordering over annotations:

Definition 4.2.8 *For annotations, write* $b <_{\mathsf{B}} c$ *if and only if* $b \leq_{\mathsf{B}} c$ *and* $b \neq c$. *Likewise, define* $c >_{\mathsf{B}} b$ *if and only if* $b <_{\mathsf{B}} c$.

Theorem 4.2.9 (Strong Non-deterministic Non-Interference) *For any valid deduction* $\Gamma \vdash^{b}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}$ *in System* $\mathcal{T}_{\mathsf{FO}'\pi\leq}$, *then for each trace* $\langle \Theta, \Gamma, \mathrm{P} \rangle \xrightarrow{\vec{\mu}} \langle \Theta', \Gamma', \mathrm{P}' \rangle \Vdash_{\mathrm{certify}} \mathbb{R}$ *the trace obtained from* $\vec{\mu}$ *by deleting all labels with integrity* c, *for* $c <_{\mathsf{B}} b$, *is also a valid trace.*

Proof. By Corollary 4.2.7. □

4.3 Safety of System $\mathcal{T}_{\mathsf{HO}\pi\leq}$

Reflecting the extra complexity of the rules of system $\mathcal{T}_{\mathsf{HO}\pi\leq}$, the proofs of type soundness are also rather more complicated (this is primarily due to the extra side-conditions required).

Before tackling even the substitution lemma, some additional results concerning contexts will be required.

4.3.1 Properties of Contexts

The following lemmata are (mostly) trivial properties of contexts that facilitate proofs of properties such as subject reduction.

Lemma 4.3.1 (Context Subset)

$$\forall \mathbb{C}_\epsilon, \mathbb{C}' \subseteq \mathbb{C}. \bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}_{\mathrm{T}}(x) \leq_{\mathsf{B}} \bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}'_{\mathrm{T}}(x)$$

Proof. Let $\overrightarrow{y}$ be the sequence of names given by $\mathbb{C}_\epsilon \cap \mathrm{dom}\mathbb{C}$. Then since $\mathbb{C}' \subseteq \mathbb{C}$ the sequence $\overrightarrow{y'} = \mathbb{C}_\epsilon \cap \mathrm{dom}\mathbb{C}'$ is a subsequence of $\overrightarrow{y}$, and hence by definition $\bigwedge_{x\in\mathbb{C}_\epsilon} \mathbb{C}_{\mathrm{T}}(x) \leq_{\mathsf{B}} \bigwedge_{x\in\mathbb{C}_\epsilon} \mathbb{C}'_{\mathrm{T}}(x)$. (since $\forall c, b \leq_{\mathsf{B}} c.b \cdot c \leq_{\mathsf{B}} c$, and where $\bigwedge_{x\in\mathbb{C}_\epsilon} \mathbb{C}_{\mathrm{T}}(x)$ and $\bigwedge_{x\in\mathbb{C}_\epsilon} \mathbb{C}'_{\mathrm{T}}(x)$ are the sequences $\bigwedge \overrightarrow{b}$ and $\bigwedge \overrightarrow{b'}$, corresponding to the mappings of $\overrightarrow{y}$ and $\overrightarrow{y'}$ in $\mathbb{C}$ and $\mathbb{C}'$ respectively). □

Corollary 4.3.2 (Context Expansion)

$$\forall \mathbb{C}, b, y \notin \mathrm{dom}\mathbb{C}.b \leq_{\mathsf{B}} \bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}_{\mathrm{T}}(x) \Rightarrow b \leq_{\mathsf{B}} \bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}'_{\mathrm{T}}(x)$$

where $\mathbb{C}' = \mathbb{C}, y : b$.

Proof. Let $\bigwedge_{x\in\mathbb{C}_\epsilon} \mathbb{C}_{\mathrm{T}}(x) = c$ (and from the statement, $b \leq_{\mathsf{B}} c$). Then given $\mathbb{C}' = \mathbb{C}, y : b$ ($y \notin \mathrm{dom}\mathbb{C}$)

$$\bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}'_{\mathrm{T}}(x) = \begin{cases} c & , \text{ if } y \notin \mathbb{C}_\epsilon \\ b \cdot c & , \text{ if } y \in \mathbb{C}_\epsilon \end{cases}$$

and since $b \leq_{\mathsf{B}} c \Rightarrow b \leq_{\mathsf{B}} b \cdot c$ then $b \leq_{\mathsf{B}} \bigwedge_{x\in\mathbb{C}_\epsilon} \mathbb{C}'_{\mathrm{T}}(x)$ as required. □

The next corollary follows straight-forwardly from the previous result:

Corollary 4.3.3

$$\forall b, c \geq_{\mathsf{B}} b, \mathbb{C}, y \notin \mathrm{dom}\mathbb{C}.b = \bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}_{\mathrm{T}}(x) \Rightarrow b = \bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}'_{\mathrm{T}}(x)$$

where $\mathbb{C}' = \mathbb{C}, y : c$.

Lemma 4.3.4 (Context Join)

$$\forall \mathbb{C}_1 \asymp \mathbb{C}_2. \bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}_{1\mathrm{T}}(x) = b \wedge \bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}_{2\mathrm{T}}(x) = c \Rightarrow \bigwedge_{x \in \mathbb{C}_\epsilon} (\mathbb{C}_1, \mathbb{C}_2)_{\mathrm{T}}(x) = b \cdot c$$

Proof. By definition of the comma operator and straightforward induction on the contents of $\mathbb{C}_1$, $\mathbb{C}_2$. □

Corollary 4.3.5 $\forall \mathbb{C}_1 \asymp \mathbb{C}_2$.

$$b_1 \leq_{\mathsf{B}} \bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}_{1\mathrm{T}}(x) \wedge b_2 \leq_{\mathsf{B}} \bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}_{2\mathrm{T}}(x) \Rightarrow b_1 \cdot b_2 \leq_{\mathsf{B}} \bigwedge_{x \in \mathbb{C}_\epsilon} (\mathbb{C}_1, \mathbb{C}_2)_{\mathrm{T}}(x)$$

Proof. Note that by Lemma 4.3.4 this is equivalent to showing that given $b_1 \leq_{\mathsf{B}} c_1$ and $b_2 \leq_{\mathsf{B}} c_2$ then $b_1 \cdot b_2 \leq_{\mathsf{B}} c_1 \cdot c_2$ is true, and by Definition 3.4.2 $b \leq_{\mathsf{B}} c \iff b \cdot c = b$, so:

$$\begin{aligned} b_1 \cdot b_2 \cdot c_1 \cdot c_2 &= b_1 \cdot c_1 \cdot b_2 \cdot c_2 \\ &= (b_1 \cdot c_1) \cdot (b_2 \cdot c_2) \\ &= b_1 \cdot b_2 \end{aligned}$$

as required. □

Lemma 4.3.6 *If $c \leq_{\mathsf{B}} c'$ and $d \leq_{\mathsf{B}} d'$, then $b \cdot c + \overline{b} \cdot d \leq_{\mathsf{B}} b \cdot c' + \overline{b} \cdot d'$.*

Proof. By Definition 3.4.2, to show $b \cdot c + \overline{b} \cdot d \leq_{\mathsf{B}} b \cdot c' + \overline{b} \cdot d'$ it is sufficient to show that $(b \cdot c + \overline{b} \cdot d) \cdot (b \cdot c' + \overline{b} \cdot d') = b \cdot c + \overline{b} \cdot d$:

$$\begin{aligned} &(b \cdot c + \overline{b} \cdot d) \cdot (b \cdot c' + \overline{b} \cdot d') \\ &\quad = b \cdot c \cdot (b \cdot c' + \overline{b} \cdot d') + \overline{b} \cdot d \cdot (b \cdot c' + \overline{b} \cdot d') \\ &\quad = b \cdot c \cdot b \cdot c' + b \cdot c \cdot \overline{b} \cdot d' + \overline{b} \cdot d \cdot b \cdot c' + \overline{b} \cdot d \cdot \overline{b} \cdot d' \\ &\quad = b \cdot b \cdot c \cdot c' + b \cdot \overline{b} \cdot c \cdot d' + \overline{b} \cdot b \cdot d \cdot c' + \overline{b} \cdot \overline{b} \cdot d \cdot d' \\ &\quad = b \cdot c \cdot c' + \overline{b} \cdot d \cdot d' \\ &\quad = b \cdot c + \overline{b} \cdot d \end{aligned}$$

as required (recalling that $c \cdot c' = c$ and $d \cdot d' = d$ by the statement). □

Lemma 4.3.7 *If $c' \leq_{\mathsf{B}} \bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}'_{\mathrm{T}}(x)$ and $c'' \leq_{\mathsf{B}} \bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}''_{\mathrm{T}}(x)$ then $b \cdot c' + \overline{b} \cdot c'' \leq_{\mathsf{B}}$ $\bigwedge_{x \in \mathbb{C}_\epsilon} (\mathbb{C}' \langle b \rangle \mathbb{C}'')_{\mathrm{T}}(x)$.*

Proof. First note that by simple manipulation

$$(b \cdot c_1 + \overline{b} \cdot d_1) \cdot (b \cdot c_2 + \overline{b} \cdot d_2) = b \cdot c_1 \cdot c_2 + \overline{b} \cdot d_1 \cdot d_2$$

and thus by extension

$$(b \cdot c_1 + \overline{b} \cdot d_1) \ldots (b \cdot c_n + \overline{b} \cdot d_n) = b \cdot c_1 \ldots c_n + \overline{b} \cdot d_1 \ldots d_n$$

From this observation and Definition 3.6.3 it can be observed that $\bigwedge_{x\in\mathbb{C}_\epsilon}(\mathbb{C}_1 \langle b\rangle \mathbb{C}_2)_{\mathrm{T}}(x) = b \cdot \bigwedge_{x\in\mathbb{C}_\epsilon}\mathbb{C}_{1\mathrm{T}}(x) + \overline{b} \cdot \bigwedge_{x\in\mathbb{C}_\epsilon}\mathbb{C}_{2\mathrm{T}}(x)$. Now, by definition of $\leq_{\mathsf{B}}$ to show that the statement holds it is necessary to show that $(b\cdot c' + \overline{b}\cdot c'')\cdot(b\cdot \bigwedge_{x\in\mathbb{C}_\epsilon}\mathbb{C}_{1\mathrm{T}}(x) + \overline{b}\cdot \bigwedge_{x\in\mathbb{C}_\epsilon}\mathbb{C}_{2\mathrm{T}}(x)) = (b\cdot c' + \overline{b}\cdot c'')$, so:

$$\begin{aligned}
&(b\cdot c' + \overline{b}\cdot c'')\cdot(b\cdot \bigwedge_{x\,\in\,\mathbb{C}_\epsilon}\mathbb{C}_{1\mathrm{T}}(x) + \overline{b}\cdot \bigwedge_{x\,\in\,\mathbb{C}_\epsilon}\mathbb{C}_{2\mathrm{T}}(x))\\
&\quad= \; b\cdot c'\cdot(b\cdot \bigwedge_{x\,\in\,\mathbb{C}_\epsilon}\mathbb{C}_{1\mathrm{T}}(x) + \overline{b}\cdot \bigwedge_{x\,\in\,\mathbb{C}_\epsilon}\mathbb{C}_{2\mathrm{T}}(x)) +\\
&\qquad\quad \overline{b}\cdot c''\cdot(b\cdot \bigwedge_{x\,\in\,\mathbb{C}_\epsilon}\mathbb{C}_{1\mathrm{T}}(x) + \overline{b}\cdot \bigwedge_{x\,\in\,\mathbb{C}_\epsilon}\mathbb{C}_{2\mathrm{T}}(x))\\
&\quad= \; b\cdot c'\cdot \bigwedge_{x\,\in\,\mathbb{C}_\epsilon}\mathbb{C}_{1\mathrm{T}}(x) + \overline{b}\cdot c''\cdot \bigwedge_{x\,\in\,\mathbb{C}_\epsilon}\mathbb{C}_{2\mathrm{T}}(x)\\
&\quad= \; b\cdot c' + \overline{b}\cdot c''
\end{aligned}$$

as required. □

Lemma 4.3.8 *If* $\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b;\mathbb{C}$ *and* $\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overline{x}[\overrightarrow{K}].\mathrm{P} : \mathsf{Proc}^b;\mathbb{C}'$ *then* $\mathbb{C}\subseteq\mathbb{C}'$.

Proof. By induction on the derivation of $\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b;\mathbb{C}$ and by the output rule. □

Proposition 4.3.9 (Context Annotation)

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b;\mathbb{C} \Rightarrow b \leq_{\mathsf{B}} \bigwedge_{x\,\in\,\mathbb{C}_\epsilon}\mathbb{C}_{\mathrm{T}}(x)$$

Proof. By induction on the derivation of $\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b;\mathbb{C}$ (note that no case is needed for the Final rule).

- Zero: Suppose the last rule used in the deduction was the zero axiom, with a consequent
$$\emptyset \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbf{0}^b : \mathsf{Proc}^b;\emptyset$$
Then clearly by definition $\mathbb{C}=\emptyset$ and hence $\bigwedge_{x\in\mathbb{C}_\epsilon}\emptyset_{\mathrm{T}}(x) = \mathrm{T}$, and $b \leq_{\mathsf{B}} \mathrm{T}$ for all b as required.

- Judgement Conversion: Trivial, by the induction hypothesis.

- Weaken: Suppose the last rule used was an instance of the weaken rule:

$$\frac{\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{A} : \sigma^b; \mathbb{C} \quad \vdash_{\mathrm{HO}\leq} \sigma'^c \quad V \notin \mathrm{dom}\Gamma}{\Gamma, V : \sigma'^c \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{A} : \sigma^b; \mathbb{C}}$$

  By the induction hypothesis the result holds in the antecedent, that is $b \leq_{\mathsf{B}} \bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}_{\mathrm{T}}(x)$. Hence noting that the context is unchanged the result holds in the consequent as well.

- Name Introduction: Suppose the last rule used was an instance of the var-1 axiom:

$$\frac{}{x : \sigma^b \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} x : \sigma^b; \emptyset}$$

  Then the result holds trivially as $\bigwedge_{x \in \mathbb{C}_\epsilon} \emptyset_{\mathrm{T}}(x) = \mathrm{T}$ and $b \leq_{\mathsf{B}} \mathrm{T}$ for all b as required.

- Variable Introduction: Suppose the last rule used was an instance of the var-2 axiom:

$$\frac{}{X : \sigma^b \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} X : \sigma^b; X : b}$$

  Then $\mathbb{C} = \{X : b\}$, so either $\bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}_{\mathrm{T}}(x) = \mathrm{T}$ if $X \notin \mathbb{C}_\epsilon$, or $\bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}_{\mathrm{T}}(x) = b$ otherwise, so either $b \leq_{\mathsf{B}} \mathrm{T}$ or $b \leq_{\mathsf{B}} b$ (by reflexivity of $\leq_{\mathsf{B}}$) for all b as required.

- Replication: Suppose the deduction ended in a use of the replication rule:

$$\frac{\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C}}{\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} !\mathrm{P} : \mathsf{Proc}^b; \mathbb{C}}$$

  By the induction hypothesis the result holds for the antecedent, and hence noting the context doesn't change the result holds in the consequent as well.

- Restriction: Suppose the last rule used was an instance of the restriction rule:

$$\frac{\Gamma_x, x : \sigma^c \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C} \quad x \notin \mathbb{C}_\epsilon}{\Gamma_x \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} (\nu x : \sigma^c)\mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_x}$$

  By the induction hypothesis the result holds for the antecedent, and hence by the Context Subset Lemma (Lemma 4.3.1) the result holds for the consequent as well by transitivity of $\leq_{\mathsf{B}}$. (Alternatively, note the requirement that $x \notin \mathbb{C}_\epsilon$ so from the perspective of $\mathbb{C}_\epsilon$ the context is unchanged, and result holds trivially).

- Composition: Suppose the last rule used was an instance of the composition rule:

$$\frac{\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_1 \quad \Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{Q} : \mathsf{Proc}^c; \mathbb{C}_2}{\Gamma, \Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P}|\mathrm{Q} : \mathsf{Proc}^{bc}; \mathbb{C}_1, \mathbb{C}_2}$$

  By the induction hypothesis the result holds for each antecedent individually, that is $b \leq_{\mathsf{B}} \bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}_{1\mathrm{T}}(x)$ and $c \leq_{\mathsf{B}} \bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}_{2\mathrm{T}}(x)$. Then by Corollary 4.3.5 $b \cdot c \leq_{\mathsf{B}} \bigwedge_{x \in \mathbb{C}_\epsilon} (\mathbb{C}_1, \mathbb{C}_2)_{\mathrm{T}}(x)$ as required.

- Summation: Similarly.

- Abstraction: Suppose the last rule used in the deduction was an instance of the abstraction rule:

$$\frac{\Gamma_{\overrightarrow{V}}, \overrightarrow{V} : \overrightarrow{\sigma^c} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C} \quad \forall V \in \overrightarrow{V}.\mathbb{C}_{\mathrm{T}}(V) \geq_{\mathsf{B}} b, V \notin \mathbb{C}_\epsilon}{\Gamma_{\overrightarrow{V}} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} (\overrightarrow{V})\mathrm{P} : (\overrightarrow{\sigma^c}) \to \mathsf{Proc}^b; \mathbb{C}_{\overrightarrow{V}}}$$

  By the induction hypothesis the result holds for the antecedent; $b \leq_{\mathsf{B}} \bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}_{\mathrm{T}}(x)$. Hence by the Context Subset Lemma (Lemma 4.3.1) and transitivity of $\leq_{\mathsf{B}}$ the result holds for the consequent as well.

- Application: Suppose the last rule used was an instance of the application rule:

$$\frac{\begin{array}{c}\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{A} : (\overrightarrow{\sigma^c}) \to \mathsf{Proc}^b; \mathbb{C}_1 \quad \overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma^d}; \overrightarrow{\mathbb{C}_2} \quad \overrightarrow{\sigma^d} \leq \overrightarrow{\sigma^c} \\ \forall \mathbb{C}_{2i}, V \in \mathrm{dom}\mathbb{C}_{2i}.\mathbb{C}_{2i}(V) \leq_{\mathsf{B}} d_i \quad \forall x \in \mathbb{C}_\epsilon.\overrightarrow{\mathbb{C}_{2\mathrm{T}}}(x) \geq_{\mathsf{B}} b\end{array}}{\Gamma, \overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{F}\langle\overrightarrow{K}\rangle : \mathsf{Proc}^b; \mathbb{C}_1, \overrightarrow{\mathbb{C}_2}, \mathsf{chans}\left(\overrightarrow{K}\right) : b}$$

  By the induction hypothesis the result holds for the first antecedent, $b \leq_{\mathsf{B}} \bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}_{1\mathrm{T}}(x)$. Then, noting the clause

$$\forall x \in \mathbb{C}_\epsilon.\overrightarrow{\mathbb{C}_{2\mathrm{T}}}(x) \geq_{\mathsf{B}} b$$

  the Context Expansion Corollary (4.3.3) applies so result holds in the consequent as well.

- Input: Suppose the last rule used was an instance of the input rule:

$$\frac{\begin{array}{c}\Gamma_{\overrightarrow{V}}, x : (\overrightarrow{\sigma^c})^b, \overrightarrow{V} : \overrightarrow{\sigma^{c'}} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^d; \mathbb{C} \\ \overrightarrow{\sigma^c} \leq \overrightarrow{\sigma^{c'}} \quad \forall V \in \overrightarrow{V}.\mathbb{C}_{\mathrm{T}}(V) \geq_{\mathsf{B}} d, V \notin \mathbb{C}_\epsilon\end{array}}{\Gamma_{\overrightarrow{V}}, x : (\overrightarrow{\sigma^c})^b \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} x(\overrightarrow{V} : \overrightarrow{\sigma^{c'}}).\mathrm{P} : \mathsf{Proc}^d; \mathbb{C}_{\overrightarrow{V}}, x : d}$$

By the induction hypothesis the result holds for the antecedent, so

$$d \leq_{\mathsf{B}} \bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}_{\mathrm{T}}(x)$$

Then, noting that the context in the consequent is a subset of that in the antecedent with an additional name at trustedness d, by the Context Subset Lemma (4.3.1) and the Context Expansion Corollary (4.3.3) the result holds in the consequent as required.

- Output: Suppose the last rule used was an instance of the output rule:

$$\frac{\begin{array}{c}\Gamma, x : (c \cdot \overrightarrow{\sigma^d})^b \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^c; \mathbb{C}_1 \quad \overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma^{d'}}; \overrightarrow{\mathbb{C}_2} \\ \overrightarrow{\sigma^{d'}} \leq c \cdot \overrightarrow{\sigma^d} \quad \forall x \in \mathbb{C}_\epsilon . \overrightarrow{\mathbb{C}_{2\mathrm{T}}}(x) \geq_{\mathsf{B}} c \quad \forall \mathbb{C}_{2i}, V \in \mathrm{dom}\mathbb{C}_{2i} . \mathbb{C}_{2i}(V) \leq_{\mathsf{B}} d'_i\end{array}}{\Gamma, x : (c \cdot \overrightarrow{\sigma^d})^b, \overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \overline{x}[\overrightarrow{K}].\mathrm{P} : \mathsf{Proc}^c; \mathbb{C}_1, x : c, \overrightarrow{\mathbb{C}_2}, \mathsf{chans}\left(\overrightarrow{K}\right) : c}$$

By the induction hypothesis the result holds for the first antecedent; that is $c \leq_{\mathsf{B}} \bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}_{1\mathrm{T}}(x)$. Then, noting the clause in the output rule $\forall x \in \mathbb{C}_\epsilon . \overrightarrow{\mathbb{C}_{2\mathrm{T}}}(x) \geq_{\mathsf{B}} c$ (for when $K \in \overrightarrow{K}$ is an agent) and the context expansion lemma (4.3.3, for when it is a name), the result holds in the consequent as well.

- Certify: Suppose the last rule used was an instance of the certify rule:

$$\frac{\begin{array}{l}\Gamma_V, V : \sigma^{\mathrm{T}}, x : (\sigma^i)^e \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^c; \mathbb{C}_1 \quad \mathbb{C}_{1\mathrm{T}}(V) \geq_{\mathsf{B}} c \\ \Theta_V, V : \sigma^{\mathrm{U}}, x : (\sigma^i)^e \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}<} \mathrm{Q} : \mathsf{Proc}^d; \mathbb{C}_2 \quad \mathbb{C}_{2\mathrm{T}}(V) \geq_{\mathsf{B}} d \quad V \notin \mathbb{C}_\epsilon\end{array}}{\begin{array}{r}\Gamma_V, \Theta_V, x : (\sigma^i)^e \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} x(V)_{\mathrm{certify}}?\mathrm{P} \oplus \mathrm{Q} : \mathsf{Proc}^{i \cdot c + \overline{i} \cdot d}; \\ \mathbb{C}_{1V} \langle i \rangle \mathbb{C}_{2V}, x : (i \cdot c + \overline{i} \cdot d)\end{array}}$$

By the induction hypothesis the result holds for both antecedents; that is $c \leq_{\mathsf{B}} \bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}_{1\mathrm{T}}(x)$ and $d \leq_{\mathsf{B}} \bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}_{2\mathrm{T}}(x)$. Then by the Context Subset Lemma (4.3.1) and Lemma 4.3.7 the result holds for the consequent as required.

□

4.3.2 Substitution Lemma

The substitution lemma may now be presented, and is considerably more complicated than those of systems $\mathcal{T}_{\mathsf{FO}\pi}$ or $\mathcal{T}_{\mathsf{FO}\pi\leq}$. Most of the side conditions, however, simply reflect the conditions imposed by the input and output rules

and the subject reduction theorem will guarantee that all such clauses do in fact hold.

The general intent of the statement is naturally similar; modulo subtyping, a variable may be replaced with one of a similar type whilst preserving well-formedness of the process. Note that because a variable may be replaced with a process as well as a name, a separate clause is necessary to introduce the new object. The portion of the statement $\mathbb{C}'_2 \subseteq \mathbb{C}_2$ in general states that if no substitution occurs (that is, $V \notin FN(\mathrm{P})$), then the statement still holds. By Lemma 4.3.1 this only affects the overall annotation on the process in a positive manner.

The various other clauses perhaps appear intimidating, but merely exist to preserve the well-formedness of the result (taking into account the external context and final rule, and associated factors). The subject reduction theorem will guarantee that any instance of a substitution will in fact satisfy these constraints as a natural consequence of the input and output rules.

The reason for the clause $\mathbb{C}'_{2\mathsf{chans}(K)} \subseteq \mathbb{C}_2$ is that if K is an agent then $\mathbb{C}'_2 \subseteq \mathbb{C}_2$, but if K is a name then $\mathbb{C}_2$ is empty yet $\mathbb{C}'_2$ may contain K if it appears in the new term, but did not exist prior to substitution.

Lemma 4.3.10 (Substitution Lemma) *If there exist valid deductions*

$$\Gamma_V, V : \sigma^b \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^c; \mathbb{C}_1 \quad \text{and} \quad \Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} K : \sigma^{b'}; \mathbb{C}_2$$

such that $\sigma^{b'} \leq \sigma^b$, $\mathbb{C}_1 \asymp \mathbb{C}_2$, $\Gamma \asymp \Theta$, *and the following conditions all hold:*

- $\forall x \in \mathsf{chans}(K) . \mathbb{C}_{1\mathrm{T}}(x) \geq_{\mathsf{B}} c$
- $\forall x \in \mathbb{C}_\epsilon . \mathbb{C}_{2\mathrm{T}}(x) \geq_{\mathsf{B}} c$
- $\forall V \in \mathrm{dom}\mathbb{C}_2 . \mathbb{C}_2(V) \leq_{\mathsf{B}} b'$
- $\mathbb{C}_{1\mathrm{T}}(V) \geq_{\mathsf{B}} c$

then for some $\mathbb{C}'_2$ *such that* $\mathbb{C}'_{2\mathsf{chans}(K)} \subseteq \mathbb{C}_2$ *there is also a valid deduction*

$$\Gamma_V, \Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P}\{K/V\} : \mathsf{Proc}^c; \mathbb{C}_{1V}, \mathbb{C}'_2$$

Proof. By induction on the derivation of $\Gamma_V, V : \sigma^b \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^c; \mathbb{C}_1$. As before, explicit reference to the conversion rule is ignored, and the statement assumed to implicitly refer to the appropriate form of deduction.

- Null Process: Suppose that the last rule used in the deduction was the null axiom:

$$\frac{}{\emptyset \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \mathbf{0}^b : \mathsf{Proc}^b; \emptyset}$$

  The result follows trivially, noting that $\mathbb{C}'_2 = \emptyset$.

- Weakening: Suppose that the last rule used in the deduction was an instance of the weaken rule:

$$\frac{\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{A} : \sigma^b; \mathbb{C}_1 \quad \vdash_{\mathrm{HO}\leq} \sigma'^c \quad W \notin \mathrm{dom}\Gamma}{\Gamma, W : \sigma'^c \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{A} : \sigma^b; \mathbb{C}}$$

  By the induction hypothesis the result holds for the antecedent; that is

$$\Gamma_V, \Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{A}\{K/V\} : \sigma^b; \mathbb{C}_{1V}, \mathbb{C}'_2$$

  Hence by the weaken rule

$$\Gamma_V, W : \sigma'^c \Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{A}\{K/V\} : \sigma^b; \mathbb{C}_{1V}, \mathbb{C}'_2$$

  as required.

- Variable Introduction: Suppose that the last rule used in the deduction was an instance of the var-2 rule:

$$\frac{\vdash_{\mathrm{HO}\leq} \sigma^b}{X : \sigma^b \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} X : \sigma^b; X : b}$$

  There are then two possibilities to consider; $V = X$ and $V \neq X$. The latter holds trivially, noting that $\mathbb{C}'_2 = \emptyset$. For the former, the following then holds

$$\Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} K : \mathsf{Proc}^c; \mathbb{C}_2$$

  where $\sigma^{b'} = \mathsf{Proc}^c$ by the statement and definition of subtyping as required, noting that $\Gamma_x = \emptyset$ and $\mathbb{C}_2 \subseteq \mathbb{C}_2$.

  Note that the var-1 rule doesn't apply in this case, as P is a process by the statement.

- Replication: By the induction hypothesis on the antecedent to the replication rule, then a use of the replication rule.

- Restriction: Suppose that the last rule used in the deduction was an instance of the restriction rule:

$$\frac{\Gamma_x, x : \sigma^c \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_1 \quad x \notin \mathbb{C}_\epsilon}{\Gamma_x \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} (\nu x : \sigma^c)\mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_{1x}}$$

  By the induction hypothesis the result holds for the antecedent:

$$\Gamma_x, x : \sigma^c, \Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P}\{K/V\} : \mathsf{Proc}^b; \mathbb{C}_{1V}, \mathbb{C}'_2$$

Then by the restriction rule

$$\Gamma_x, \Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} (\nu x : \sigma^c)\mathrm{P}\{K/V\} : \mathsf{Proc}^b; \mathbb{C}_{1xV}, \mathbb{C}'_2$$

as required, noting that $x \neq K, V$ by the statement and assumption of Barendregt's convention.

- Composition: Suppose that the last rule used in the deduction was an instance of the composition rule:

$$\frac{\Gamma_1 \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C}'_1 \quad \Gamma_2 \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{Q} : \mathsf{Proc}^c; \mathbb{C}''_1}{\Gamma_1, \Gamma_2 \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P}|\mathrm{Q} : \mathsf{Proc}^{bc}; \mathbb{C}'_1, \mathbb{C}''_1}$$

By the induction hypothesis the result holds for the antecedents individually; that is

$$\begin{array}{rrl} & \Gamma_{1V}, \Theta & \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \quad \mathrm{P}\{K/V\} : \mathsf{Proc}^b; \mathbb{C}'_{1V}, \mathbb{C}'_2 \\ \text{and} & \Gamma_{2V}, \Theta & \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \quad \mathrm{Q}\{K/V\} : \mathsf{Proc}^c; \mathbb{C}''_{1V}, \mathbb{C}''_2 \end{array}$$

Hence by the composition rule and the definition of substitution

$$\Gamma_{1V}, \Gamma_{2V}, \Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} (\mathrm{P}|\mathrm{Q})\{K/V\} : \mathsf{Proc}^{bc}; \mathbb{C}'_{1V}, \mathbb{C}''_{1V}, \mathbb{C}'_2, \mathbb{C}''_2$$

noting that $\mathbb{C}'_2 \subseteq \mathbb{C}_2$ and $\mathbb{C}''_2 \subseteq \mathbb{C}_2$ imply $\mathbb{C}'_2, \mathbb{C}''_2 \subseteq \mathbb{C}_2$ as required.

- Summation: Similarly.

- Abstraction: Suppose that the last rule used in the deduction was an instance of the abstraction rule:

$$\frac{\Gamma_{\overrightarrow{W}}, \overrightarrow{W} : \overrightarrow{\sigma^c} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_1 \quad \forall W \in \overrightarrow{W}.\mathbb{C}_{\mathrm{T}}(W) \geq_{\mathsf{B}} b, W \notin \mathbb{C}_\epsilon}{\Gamma_{\overrightarrow{W}} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} (\overrightarrow{W})\mathrm{P} : (\overrightarrow{\sigma^c}) \to \mathsf{Proc}^b; \mathbb{C}_{1\overrightarrow{W}}}$$

By the induction hypothesis the result holds for the antecedent:

$$\Gamma_{V\overrightarrow{W}}, \overrightarrow{W} : \overrightarrow{\sigma^c}, \Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P}\{K/V\} : \mathsf{Proc}^b; \mathbb{C}_{1V}, \mathbb{C}'_2$$

Hence by the abstraction rule

$$\Gamma_{V\overrightarrow{W}}\Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} (\overrightarrow{W} : \overrightarrow{\sigma^c})(\mathrm{P}\{K/V\}) : (\overrightarrow{\sigma^c}) \to \mathsf{Proc}^b; \mathbb{C}_{1V\overrightarrow{W}}, \mathbb{C}'_2$$

as required, noting that $V \notin \overrightarrow{W}$ and $\forall W \in \overrightarrow{W}.W \notin FN(K)$ by the statement and the assumption of Barendregt's convention.

- Application: Suppose that the last rule used in the deduction was an instance of the application rule:

$$\frac{\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{A} : (\overrightarrow{\sigma^c}) \to \mathsf{Proc}^b; \mathbb{C}'_1 \quad \overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{J} : \overrightarrow{\sigma^d}; \overrightarrow{\mathbb{C}''_1} \quad \overrightarrow{\sigma^d} \leq \overrightarrow{\sigma^c} \quad \forall \mathbb{C}''_{1i}, V \in \mathrm{dom}\mathbb{C}''_{1i}.\mathbb{C}''_{1i}(V) \leq_{\mathsf{B}} d_i \quad \forall x \in \mathbb{C}_\epsilon.\overrightarrow{\mathbb{C}''_{1\mathrm{T}}}(x) \geq_{\mathsf{B}} b}{\Gamma, \overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{A}\langle \overrightarrow{J} \rangle : \mathsf{Proc}^b; \mathbb{C}'_1, \overrightarrow{\mathbb{C}''_1}, \mathsf{chans}\left(\overrightarrow{K}\right) : b}$$

  By the induction hypothesis the result holds for the first antecedent:

$$\Gamma_V, \Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{A}\{K/V\} : (\overrightarrow{\sigma^c}) \to \mathsf{Proc}^b; \mathbb{C}'_{1V}, \mathbb{C}'_2$$

  Similarly, by the induction hypothesis the result also holds for the second antecedents (considered *en masse* for convenience):

$$\overrightarrow{\Gamma'_V}, \Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{J}\{K/V\} : \overrightarrow{\sigma^d}; \overrightarrow{\mathbb{C}''_{1V}}, \mathbb{C}''_2$$

  Then by the application type rule and the definition of substitution, the result holds for the consequent as well:

$$\Gamma_V, \overrightarrow{\Gamma'_V}, \Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} (\mathrm{F}\langle \overrightarrow{J} \rangle)\{K/V\} : \mathsf{Proc}^b; \mathbb{C}'_{1V}, b \cdot \overrightarrow{c} \odot \overrightarrow{\mathbb{C}''_{1V}}, \mathsf{chans}\left(\overrightarrow{J}\{K/V\}\right) : b, \mathbb{C}'_2, \mathbb{C}''_2$$

  Note that all conditions for the application rule are satisfied:

  - $\mathbb{C}'_2 \subseteq \mathbb{C}_2$ and $\mathbb{C}''_2 \subseteq \mathbb{C}_2$ implies $\mathbb{C}'_2 \cup \mathbb{C}''_2 \subseteq \mathbb{C}_2$
  - $\forall x \in \mathbb{C}_\epsilon.(\overrightarrow{\mathbb{C}''_{1V}, \mathbb{C}''_2})_{\mathrm{T}}(x) \geq_{\mathsf{B}} b$ and $\forall \mathbb{C}''_{1i}, V \in \mathrm{dom}\mathbb{C}''_{1i}.(\mathbb{C}''_{1i}, \mathbb{C}''_2)(V) \leq_{\mathsf{B}} d_i$ both hold by the statement.

- Input: Suppose that the last rule used in the deduction was an instance of the input rule:

$$\frac{\Gamma_{\overrightarrow{W}}, x : (\overrightarrow{\sigma^c})^b, \overrightarrow{W} : \overrightarrow{\sigma^{c'}} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^d; \mathbb{C}_1 \quad \overrightarrow{\sigma^c} \leq \overrightarrow{\sigma^{c'}} \quad \forall W \in \overrightarrow{W}.\mathbb{C}_1(W) \geq_{\mathsf{B}} d, W \notin \mathbb{C}_\epsilon}{\Gamma_{\overrightarrow{W}}, x : (\overrightarrow{\sigma^c})^b \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} x(\overrightarrow{W} : \overrightarrow{\sigma^{c'}}).\mathrm{P} : \mathsf{Proc}^d; \mathbb{C}_{1\overrightarrow{W}}, x : d}$$

  By the induction hypothesis the result holds for the antecedent:

$$\Gamma_{V\overrightarrow{W}}, x : (\overrightarrow{\sigma^c})^b, \overrightarrow{W} : \overrightarrow{\sigma^{c'}}, \Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P}\{K/V\} : \mathsf{Proc}^d; \mathbb{C}_{1V}, \mathbb{C}'_2$$

  There are two cases to consider; $V = x$ and $V \neq x$ (note that $V \notin \overrightarrow{W}$ by the statement and the assumption of Barendregt's convention).

The latter case follows easily by the input rule and the definition of substitution:

$$\Gamma_{V\overrightarrow{W}}, x : (\overrightarrow{\sigma^c})^b, \Theta \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} (x(\overrightarrow{W} : \overrightarrow{\sigma^{c'}}).\mathrm{P})\{K/V\} : \mathsf{Proc}^d; \mathbb{C}_{1\overrightarrow{W}}, x : d, \mathbb{C}'_2$$

In the former case the result also follows reasonably easily by the input rule, modulo consideration for subtyping. Assume $V = x$ and $K = y$ for some y:

$$\Gamma_{x\overrightarrow{W}}, y : (\overrightarrow{\sigma^c})^{b'}, \Theta \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} (x(\overrightarrow{W} : \overrightarrow{\sigma^{c'}}).\mathrm{P})\{y/x\} : \mathsf{Proc}^d; \mathbb{C}_{1\overrightarrow{W}}, y : d, \mathbb{C}'_2$$

Note that by the definition of subtyping, only the outer-most annotation is affected in $(\overrightarrow{\sigma^c})^{b'} \leq (\overrightarrow{\sigma^c})^b$.

- Output: Suppose that the last rule used in the deduction was an instance of the output rule:

$$\frac{\begin{array}{c}\Gamma, x : (c \cdot \overrightarrow{\sigma^d})^b \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^c; \mathbb{C}'_1 \quad \overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma^{d'}}; \overrightarrow{\mathbb{C}''_1} \\ \overrightarrow{\sigma^{d'}} \leq c \cdot \overrightarrow{\sigma^d} \quad \forall x \in \mathbb{C}_\epsilon.\overrightarrow{\mathbb{C}''_{1\mathrm{T}}}(x) \geq_{\mathsf{B}} c \quad \forall \mathbb{C}''_{1i}, V \in \mathrm{dom}\mathbb{C}''_{1i}.\mathbb{C}''_{1i}(V) \leq_{\mathsf{B}} d'_i\end{array}}{\Gamma, x : (c \cdot \overrightarrow{\sigma^d})^b, \overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \overline{x}[\overrightarrow{K}].\mathrm{P} : \mathsf{Proc}^c; \mathbb{C}'_1, x : c, \overrightarrow{\mathbb{C}''_2}, \mathsf{chans}\Big(\overrightarrow{K}\Big) : c}$$

By the induction hypothesis the result holds for the first antecedent:

$$\Gamma_V, x : (c \cdot \overrightarrow{\sigma^d})^b, \Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P}\{K/V\} : \mathsf{Proc}^c; \mathbb{C}'_{1V}, \mathbb{C}'_2$$

where $\mathbb{C}'_{\mathsf{2chans}(K)} \subseteq \mathbb{C}_2$. Again by the induction hypothesis it also holds for the subject antecedents:

$$\overrightarrow{\Gamma'_V}, \Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{J}\{K/V\} : \overrightarrow{\sigma^{d'}}; \overrightarrow{\mathbb{C}''_{1V}}, \mathbb{C}''_2$$

where $\mathbb{C}''_{\mathsf{2chans}(K)} \subseteq \mathbb{C}_2$ and by the statement $\forall x \in \mathbb{C}_\epsilon.(\overrightarrow{\mathbb{C}''_1, \mathbb{C}''_2})_{\mathrm{T}}(x) \geq_{\mathsf{B}} c$ and $\forall \mathbb{C}''_{1i}, V \in \mathrm{dom}(\mathbb{C}''_{1i}, \mathbb{C}''_2).(\mathbb{C}''_{1i}, \mathbb{C}''_2)(V) \leq_{\mathsf{B}} d'_i$. There are now two cases to consider before the output rule can be applied; $V = x$ and $V \neq x$. The latter follows trivially by the output rule:

$$\begin{array}{l}\Gamma_V, x : (c \cdot \overrightarrow{\sigma^d})^b, \overrightarrow{\Gamma'_V}, \Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} (\overline{x}[\overrightarrow{J}].\mathrm{P})\{K/V\} : \mathsf{Proc}^c; \\ \qquad\qquad \mathbb{C}'_{1V}, x : c, \overrightarrow{\mathbb{C}''_{1V}}, \mathbb{C}''_2, \mathsf{chans}\Big(\overrightarrow{J}\{K/V\}\Big) : c, \mathbb{C}'_2\end{array}$$

If $V = x$, and $K = y$ for some y, the result follows similarly:

$$\begin{array}{l}\Gamma_x, y : (c \cdot \overrightarrow{\sigma^d})^{b'}, \overrightarrow{\Gamma'_x} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} (\overline{x}[\overrightarrow{J}].\mathrm{P})\{y/x\} : \mathsf{Proc}^c; \\ \qquad\qquad \mathbb{C}'_{1x}, y : c, \overrightarrow{\mathbb{C}''_{1x}}, \mathbb{C}''_2, \mathsf{chans}\Big(\overrightarrow{J}\{y/x\}\Big) : c, \mathbb{C}'_2\end{array}$$

where $(c \cdot \overrightarrow{\sigma^d})^{b'} \leq (c \cdot \overrightarrow{\sigma^d})^b$ by the statement, and recalling from the definition of subtyping that only the outermost annotation may differ.

- Certify: Suppose that the last rule used in the deduction was an instance of the certify rule:

$$\frac{\Gamma'_W, W : \sigma^{\mathrm{T}}, x : (\sigma^i)^e \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^c; \mathbb{C}'_1 \quad \mathbb{C}_{1\mathrm{T}}(W) \geq_{\mathsf{B}} c \quad \Gamma''_W, W : \sigma^{\mathrm{U}}, x : (\sigma^i)^e \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}<} \mathrm{Q} : \mathsf{Proc}^d; \mathbb{C}''_1 \quad \mathbb{C}_{1\mathrm{T}}(W) \geq_{\mathsf{B}} d \quad W \notin \mathbb{C}_\epsilon}{\Gamma'_W, \Theta''_W, x : (\sigma^i)^e \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} x(W : \sigma^i)_{\mathrm{certify}}?\mathrm{P} \oplus \mathrm{Q} : \mathsf{Proc}^{i\cdot c+\bar{i}\cdot d}; \mathbb{C}'_{1W} \langle i \rangle \mathbb{C}''_{1W}, x : (i \cdot c + \bar{i} \cdot d)}$$

  By the induction hypothesis the result holds for the antecedents individually; that is

$$\begin{aligned} & \Gamma'_{VW}, W : \sigma^{\mathrm{T}}, x : (\sigma^i)^e \ \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \ \mathrm{P}\{K/V\} : \mathsf{Proc}^c; \mathbb{C}'_1, \mathbb{C}'_2 \\ \text{and} \quad & \Gamma''_{VW}, W : \sigma^{\mathrm{U}}, x : (\sigma^i)^e \ \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \ \mathrm{Q}\{K/V\} : \mathsf{Proc}^d; \mathbb{C}''_1, \mathbb{C}''_2 \end{aligned}$$

  There are then two cases to consider when applying the certify rule; $V = x$ and $V \neq x$. For the latter, by the certify rule and the definition of substitution then

$$\Gamma'_{VW}, \Gamma''_{VW}, x : (\sigma^i)^e \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} (x(W : \sigma^i)_{\mathrm{certify}}?\mathrm{P} \oplus \mathrm{Q})\{K/V\} : \mathsf{Proc}^{i\cdot c+\bar{i}\cdot d}; (\mathbb{C}'_{1VW}, \mathbb{C}'_2) \langle i \rangle (\mathbb{C}''_{1VW}, \mathbb{C}''_2), x : (i \cdot c + \bar{i} \cdot d)$$

  Otherwise, if $V = x$, then assuming $K = y$ for some y, then

$$\Gamma'_{xW}, \Gamma''_{xW}, y : (\sigma^i)^{e'} \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} (x(W : \sigma^i)_{\mathrm{certify}}?\mathrm{P} \oplus \mathrm{Q})\{y/x\} : \mathsf{Proc}^{i\cdot c+\bar{i}\cdot d}; (\mathbb{C}'_{1xW}, \mathbb{C}'_2) \langle i \rangle (\mathbb{C}''_{1xW}, \mathbb{C}''_2), y : (i \cdot c + \bar{i} \cdot d)$$

  where $(\sigma^i)^{e'} \leq (\sigma^i)^e$ by the statement, recalling that by the definition of subtyping only the outermost annotation is different.

□

Corollary 4.3.11 (Substitution Corollary) *If there exist valid deductions*

$$\Gamma_{\overrightarrow{V}}, \overrightarrow{V} : \overrightarrow{\sigma^b} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^c; \mathbb{C}_1 \quad \text{and} \quad \overrightarrow{\Theta} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma^{b'}}; \overrightarrow{\mathbb{C}_2}$$

such that all $\overrightarrow{V}$ are distinct, $\overrightarrow{\sigma^{b'}} \leq \overrightarrow{\sigma^b}$, $\mathbb{C}_1 \asymp \overrightarrow{\mathbb{C}_2}$, $\Gamma \cup \overrightarrow{\Theta}$ is defined, and the following conditions all hold:

- $\forall W \in \mathsf{chans}\left(\overrightarrow{K}\right).(\mathbb{C}_1, \overrightarrow{\mathbb{C}_2})_{\mathrm{T}}(W) \geq_{\mathsf{B}} c$
- $\forall x \in \mathbb{C}_\epsilon.\overrightarrow{\mathbb{C}}_{2\mathrm{T}}(x) \geq_{\mathsf{B}} c$
- $\forall V \in \overrightarrow{V}.\mathbb{C}_{1\mathrm{T}}(V) \geq_{\mathsf{B}} c$

Then for some $\overrightarrow{\mathbb{C}'_2} \subseteq \overrightarrow{\mathbb{C}_2}$ there is also a valid deduction

$$\Gamma_{\overrightarrow{V}}, \overrightarrow{\Theta} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P}\{\overrightarrow{K}/\overrightarrow{V}\} : \mathsf{Proc}^c; \mathbb{C}_{1V}, \overrightarrow{\mathbb{C}'_2}$$

Proof. By Lemma 4.3.10 (the Substitution Lemma) above. □

4.3.3 Subject Reduction

The development of the subject reduction result is done in three steps:

- A general Lemma over the typed labelled transition semantics and accompanying generated substitution;
- A theorem as a specialisation of this Lemma to internal reductions only, with a multi-step reductions corollary; and
- A result specialising this further to deductions after application of the Final rule (that is, no contexts are present in the deduction).

As usual, in the statements the corresponding identical cases for judgements concerning guarded and sum processes are ignored, and the judgement assumed to be of the correct form for the process in question.

Lemma 4.3.12 (Subject Reduction) *If there is a valid deduction in System* $\mathcal{T}_{\mathsf{HO}\pi\leq}$ *for*

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_1$$

and a reduction

$$\langle \Theta | \mathbb{C}_2, \Gamma | \mathbb{C}_1, \mathrm{P} \rangle \xrightarrow{\mu} \langle \Theta' | \mathbb{C}'_2, \Gamma' | \mathbb{C}'_1, \mathrm{P}' \rangle \Vdash_{\text{certify}} \mathbb{R}$$

then there is also a valid deduction for

$$\mathbb{R}(\Gamma' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P}' : \mathsf{Proc}^b; \mathbb{C}'_1)$$

where $\mathbb{C}'_1 \asymp \mathbb{C}_1$ *and* $\forall x \in FN(\mu) \cap \mathbb{C}_\epsilon . x \in \mathrm{dom}\mathbb{C}'_1$ *implies* $\mathbb{R}(\mathbb{C}'_1(x)) \geq_{\mathsf{B}} \mathbb{R}(b)$.

Proof. By induction on the derivation of

$$\langle \Theta | \mathbb{C}_2, \Gamma | \mathbb{C}_1, \mathrm{P} \rangle \xrightarrow{\mu} \langle \Theta' | \mathbb{C}'_2, \Gamma' | \mathbb{C}'_1, \mathrm{P}' \rangle \Vdash_{\text{certify}} \mathbb{R}$$

The cases for alpha-congruence, application, summation, composition, replication, restriction, and communication follow directly from the induction hypothesis, and the case for open follows from the cases for output and restriction. As an outline of a representative case, consider the case for replication:

Suppose the derivation of reduction ended in a use of the replication case:

$$\frac{\langle \Theta | \mathbb{C}_2, \Gamma | \mathbb{C}_1, \mathrm{P} \rangle \xrightarrow{\mu} \langle \Theta' | \mathbb{C}'_2, \Gamma' | \mathbb{C}'_1, \mathrm{P}' \rangle \Vdash_{\text{certify}} \mathbb{R}}{\langle \Theta | \mathbb{C}_2, \Gamma | \mathbb{C}_1, !\mathrm{P} \rangle \xrightarrow{\mu} \langle \Theta' | \mathbb{C}'_2, \Gamma' | \mathbb{C}'_1 \cup \mathbb{C}_1, \mathrm{P}' | \mathbb{R}(!\mathrm{P}) \rangle \Vdash_{\text{certify}} \mathbb{R}}$$

By the induction hypothesis the result holds for the antecedent; that is there is a valid deduction

$$\mathbb{R}(\Gamma' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P}' : \mathsf{Proc}^b; \mathbb{C}'_1)$$

From the statement it is also the case that

$$\mathbb{R}(\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} !\mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_1)$$

and hence by the composition rule

$$\mathbb{R}(\Gamma' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P}'|!\mathrm{P} : \mathsf{Proc}^b; \mathbb{C}'_1, \mathbb{C}_1)$$

as required, where $\Gamma \subseteq \Gamma'$ (note that it is not necessarily the case that $\mathbb{C}_1 \subseteq \mathbb{C}'_1$, as the action may have involved transference of names in either direction). Recall that multiplication is idempotent so $b \cdot b = b$ as required.

The remaining cases for input, output, and certify require more work, but essentially stem from observing the correspondence between the pre-conditions and the associated type rules, applying the Substitution Corollary (4.3.11) where appropriate.

- Input: Suppose the last rule in the derivation was an instance of the input case:

$$\frac{\begin{array}{c}\overrightarrow{\Theta'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma^{d'}}; \overrightarrow{\mathbb{C}'_2} \quad \bigcup\overrightarrow{\Theta'} \subseteq \Theta \quad \mathrm{dom}\bigcup\overrightarrow{\Theta'} = FN\left(\overrightarrow{K}\right) \\ \forall x \in \mathbb{C}_\epsilon.\overrightarrow{\mathbb{C}'_{2\mathrm{T}}}(x) \geq_{\mathsf{B}} b \quad \forall \mathbb{C}'_{2i}, V \in \mathrm{dom}\mathbb{C}'_{2i}.\mathbb{C}'_{2i}(V) \leq_{\mathsf{B}} d'_i \\ \overrightarrow{\sigma^{d'}} \leq \overrightarrow{\sigma^d} \quad \mathbb{C}''_2 = \overrightarrow{\mathbb{C}'_2}, \mathsf{chans}\left(\overrightarrow{K}\right) : b \quad \exists \mathbb{C}'''_2 \subseteq \mathbb{C}''_2\end{array}}{\begin{array}{r}\langle\Theta|\mathbb{C}_2 \cup \mathbb{C}''_2, \Gamma|\mathbb{C}_1 \cup \{x : b\}, x(\overrightarrow{V} : \overrightarrow{\sigma^d}).\mathrm{P}\rangle \xrightarrow{x(\overrightarrow{K})} \\ \langle\Theta|\mathbb{C}_2, \Gamma \cup \overrightarrow{\Theta'}|\mathbb{C}_1 \cup \mathbb{C}'''_2, \mathrm{P}\{\overrightarrow{K}/\overrightarrow{V}\}\rangle \Vdash_{\mathrm{certify}} \mathsf{Id}\end{array}}$$

  From the statement then there is a deduction

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} x(\overrightarrow{V} : \overrightarrow{\sigma^d}).\mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_1, x : b$$

  This deduction must have ended in zero or more uses of the weaken rule, preceded by an instance of the input rule with antecedent

$$\Gamma_{1\overrightarrow{V}}, x : (\overrightarrow{\sigma^{d''}})^c, \overrightarrow{V} : \overrightarrow{\sigma^d} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{b'}; \mathbb{C}'_1$$

  where $\mathbb{C}_1 = \mathbb{C}'_{1\overrightarrow{V}}$, $\overrightarrow{\sigma^{d''}} \leq \overrightarrow{\sigma^d}$, $\forall V \in \overrightarrow{V}.\mathbb{C}'_{1\mathrm{T}}(V) \geq_{\mathsf{B}} b$ and $V \notin \mathbb{C}_\epsilon$, and assuming $\Gamma_{1\overrightarrow{V}}, x : (\overrightarrow{\sigma^{d''}})^c \subseteq \Gamma$ (by the weaken rule).

Then by the input reduction rule

$$\overrightarrow{\Theta'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma^{d'}}; \overrightarrow{\mathbb{C}'_2}$$

where $\overrightarrow{\sigma^{d'}} \leq \overrightarrow{\sigma^{d}}$, $\forall x \in \mathbb{C}_\epsilon . \overrightarrow{\mathbb{C}'_{2\mathrm{T}}}(x) \geq_{\mathsf{B}} b$, $\forall \mathbb{C}'_{2i}, V \in \mathrm{dom}\mathbb{C}'_{2i} . \mathbb{C}'_{2i}(V) \leq_{\mathsf{B}} d'_i$, and $\mathbb{C}''_2 = \overrightarrow{\mathbb{C}'_2}, \mathsf{chans}\left(\overrightarrow{K}\right) : b$. Hence by Corollary 4.3.11 (Substitution) and as many applications of the weaken rule as necessary

$$\Gamma, \overrightarrow{\Theta'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P}\{\overrightarrow{K}/\overrightarrow{V}\} : \mathsf{Proc}^b; \mathbb{C}_1, \mathbb{C}'''_2$$

as required, noting that the substitution generated is the identity substitution and $\mathbb{C}'''_2 \subseteq \mathbb{C}''_2$ is the subset of names and variables actually bound in the substitution. Note that by the transition rule, all names in $\overrightarrow{K}$ that are also in $\mathbb{C}_\epsilon$ are in a context at least as safe as b, as required by the statement.

- Output: Suppose the last rule in the derivation was an instance of the output case:

$$\frac{\begin{array}{c}\overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma^{d}}; \overrightarrow{\mathbb{C}'_1} \quad \bigcup \overrightarrow{\Gamma'} \subseteq \Gamma \quad \mathrm{dom} \bigcup \overrightarrow{\Gamma'} = FN\left(\overrightarrow{K}\right) \\ \forall x \in \mathbb{C}_\epsilon . \overrightarrow{\mathbb{C}'_{1\mathrm{T}}}(x) \geq_{\mathsf{B}} b \quad \forall \mathbb{C}'_{1i}, V \in \mathrm{dom}\mathbb{C}'_{1i} . \mathbb{C}'_{1i}(V) \leq_{\mathsf{B}} d_i \\ \mathbb{C}''_1 = \overrightarrow{\mathbb{C}'_1}, \mathsf{chans}\left(\overrightarrow{K}\right) : b \quad \exists \mathbb{C}'''_1 \subseteq \mathbb{C}''_1\end{array}}{\begin{array}{r}\langle \Theta | \mathbb{C}_2, \Gamma | \mathbb{C}_1 \cup \mathbb{C}''_1 \cup \{x : b\}, \overline{x}[\overrightarrow{K}].\mathrm{P}\rangle \xrightarrow{\overline{x}[\overrightarrow{K}]} \\ \langle \Theta \cup \overrightarrow{\Gamma'} | \mathbb{C}_2 \cup \mathbb{C}'''_1, \Gamma | \mathbb{C}_1, \mathrm{P}\rangle \Vdash_{\mathrm{certify}} \mathsf{Id}\end{array}}$$

From the statement then there is a derivation for

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \overline{x}[\overrightarrow{K}].\mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_1, \mathbb{C}''_1, x : b$$

where $\overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma^{d}}; \overrightarrow{\mathbb{C}'_2}$ (with $\bigcup \overrightarrow{\Gamma'} \subseteq \Gamma$ and $\mathrm{dom} \bigcup \overrightarrow{\Gamma'} = FN\left(\overrightarrow{K}\right)$), $\forall x \in \mathbb{C}_\epsilon . \overrightarrow{\mathbb{C}'_{1\mathrm{T}}}(x) \geq_{\mathsf{B}} b$, and $\mathbb{C}''_1 = \overrightarrow{\mathbb{C}'_1}, \mathsf{chans}\left(\overrightarrow{K}\right) : b$. This deduction must have ended in zero or more uses of the weaken rule, preceded by an instance of the output rule with antecedents

$$\Gamma_1, x : (b \cdot \overrightarrow{\sigma^{d'}})^c \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_1 \quad \text{and} \quad \overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma^{d}}; \overrightarrow{\mathbb{C}_2}$$

where $\Gamma_1, x : (b \cdot \overrightarrow{\sigma^{d'}})^c \subseteq \Gamma$, $\overrightarrow{\sigma^{d}} \leq b \cdot \overrightarrow{\sigma^{d'}}$, and $\forall x \in \mathbb{C}_\epsilon . \overrightarrow{\mathbb{C}_{2\mathrm{T}}}(x) \geq_{\mathsf{B}} b$ and $\forall \mathbb{C}_{2i}, V \in \mathbb{C}_{2i} . \mathbb{C}_{2i}(V) \leq_{\mathsf{B}} d_i$. The by as many applications of the weaken rule as necessary,

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_1$$

as required, noting that the substitution generated is the identity substitution.

- Certify (as trusted): Suppose the last rule in the derivation was an instance of the certify-as-trusted case:

$$\frac{\begin{array}{c}\Gamma(x) = (\sigma^i)^e \quad \text{certify}(K) = \mathrm{T} \quad \mathbb{R} = \mathsf{Id}[i := \mathrm{T}] \\ \Theta' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} K : \sigma^f; \mathbb{C}'_2 \quad \Theta' \subseteq \Theta \quad \mathrm{dom}\Theta' = FN(K) \\ \forall V \in \mathrm{dom}\mathbb{C}'_2.\mathbb{C}'_2(V) \leq_{\mathsf{B}} f \quad \forall x \in \mathbb{C}_\epsilon.\mathbb{C}'_{2\mathrm{T}}(x) \geq_{\mathsf{B}} b = i \cdot c + \bar{i} \cdot d \\ \sigma^f \leq \sigma^i \quad \mathbb{C}''_2 = \mathbb{C}'_2, \mathsf{chans}(K) : b \quad \exists \mathbb{C}'''_2 \subseteq \mathbb{C}''_2\end{array}}{\begin{array}{r}\langle \Theta | \mathbb{C}_2 \cup \mathbb{C}''_2, \Gamma | \mathbb{C}'_1 \ \langle i \rangle \ \mathbb{C}''_1 \cup \{x : b\}, x(V : \sigma^i)_{\mathrm{certify}}^{?} \mathrm{P} \oplus \mathrm{Q} \rangle \xrightarrow{x(K)} \\ \langle \mathbb{R}(\Theta | \mathbb{C}_2), \mathbb{R}(\Gamma \cup \Theta' | \mathbb{C}'_1 \cup \mathbb{C}'''_2), \mathbb{R}(\mathrm{P}\{K/V\}) \rangle \Vdash_{\mathrm{certify}} \mathbb{R}\end{array}}$$

From the statement there is a derivation for

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} x(V : \sigma^i)_{\mathrm{certify}}^{?} \mathrm{P} \oplus \mathrm{Q} : \mathsf{Proc}^b; \mathbb{C}'_{1V} \ \langle i \rangle \ \mathbb{C}''_{1V}, x : b$$

This deduction must have ended in zero or more uses of the weaken rule, preceded by an instance of the certify rule with antecedents

$$\begin{array}{rc} & \Gamma', V : \sigma^{\mathrm{T}}, x : (\sigma^i)^e \ \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \ \mathrm{P} : \mathsf{Proc}^c; \mathbb{C}'_1 \\ \text{and} & \Gamma'', V : \sigma^{\mathrm{U}}, x : (\sigma^i)^e \ \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \ \mathrm{Q} : \mathsf{Proc}^d; \mathbb{C}''_1 \end{array}$$

where $b = i \cdot c + \bar{i} \cdot d$, $\mathbb{C}'_{1T}(V) \geq_{\mathsf{B}} c$, and $\mathbb{C}''_{1T}(V) \geq_{\mathsf{B}} d$, and $\Gamma', \Gamma'', x : (\sigma^i)^e, \subseteq \Gamma$ (and assuming that V is not in the domain of either Γ' or Γ''). Then, given $\mathbb{R} = \mathsf{Id}[i := \mathrm{T}]$, by the Substitution Lemma (4.3.10)

$$\mathbb{R}(\Gamma', \Theta', x : (\sigma^i)^e \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P}\{K/V\} : \mathsf{Proc}^c; \mathbb{C}'_{1V}, \mathbb{C}'''_2)$$

and thus by as many applications of the weaken and subtyping rules as necessary,

$$\mathbb{R}(\Gamma, \Theta' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P}\{K/V\} : \mathsf{Proc}^b; \mathbb{C}'_{1V}, \mathbb{C}'''_2)$$

noting that $\mathbb{R}(b) = \mathbb{R}(c)$ and that every name in $\mathbb{C}_\epsilon$ that is also in the free names of K is in a context at least as trusted as b as required by the statement.

- Certify (as untrusted): Similarly.

□

The next result specialises this to internal reductions:

Theorem 4.3.13 (Internal Subject Reduction) *For all Θ, $\mathbb{C}_2$, if there is a valid deduction*

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_1$$

and a reduction

$$\langle \Theta | \mathbb{C}_2, \Gamma | \mathbb{C}_1, \mathrm{P} \rangle \xrightarrow{\tau} \langle \Theta | \mathbb{C}_2, \Gamma' | \mathbb{C}'_1, \mathrm{P}' \rangle \Vdash_{\mathrm{certify}} \mathbb{R}$$

then there is a valid derivation for

$$\mathbb{R}(\Gamma' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P}' : \mathsf{Proc}^b; \mathbb{C}'_1)$$

where $\mathrm{dom}\mathbb{C}'_1 \subseteq \mathbb{C}_1$ *and* $\forall x \in \mathbb{C}_\epsilon . \mathbb{R}(\mathbb{C}'_{1\mathrm{T}})(x) \geq_{\mathsf{B}} \mathbb{R}(b)$.

Proof. By Lemma 4.3.12. Only one case from the rules in Figures 3.9 and 3.10 introduces a τ (internal) reduction; the case for communication. This combines two reductions, one for output, and one for input. The input action can be generated either by the case for input processes, or by either of the certify cases.

Examination of the two complimentary actions, together with the composition type rule, shows that the resultant context must have a domain no larger than the initial context. The remainder of the cases follow by the induction hypothesis. □

The final related reduction safety result is perhaps the most important (or at least, most pertinent to hosts); it states that the trustedness derived from a use of the rule Final is the *least* (or most conservative) level of the process, and it will only become more safe through execution. This is a natural consequence of expiry of names; if a process is considered (after final) untrusted due to a single name shared with the host being in an untrusted context, and that name expires during reduction, then the program may be considered trusted after that reduction. The converse is never true.

As a simple example, consider the process $x.\mathbf{0}^{\mathrm{U}} | \overline{x}.\mathbf{0}^{\mathrm{U}}$ with $\mathbb{C}_\epsilon = \{x\}$. This process would have a type of $\mathsf{Proc}^{\mathrm{U}}$ after the Final rule, as x appears in an untrusted context. However, after a single internal reduction, it has reduced to $\mathbf{0}^{\mathrm{U}} | \mathbf{0}^{\mathrm{U}}$ which is now trusted by the Final rule, as no name in the external context appears in an untrusted context (or at all) any more.

Theorem 4.3.14 (Final Subject Reduction) $\exists c \geq_{\mathsf{B}} b$. *If there is a valid deduction*

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b$$

and for some Θ, $\mathbb{C}_2$, and $\mathbb{C}_1$ a reduction

$$\langle \Theta | \mathbb{C}_2, \Gamma | \mathbb{C}_1, \mathrm{P} \rangle \xrightarrow{\tau} \langle \Theta | \mathbb{C}_2, \Gamma' | \mathbb{C}'_1, \mathrm{P}' \rangle \Vdash_{\mathrm{certify}} \mathbb{R}$$

Then there is a valid deduction

$$\mathbb{R}(\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P}' : \mathsf{Proc}^{c})$$

such that $b \leq_{\mathsf{B}} c$*, and further it holds that* $b \leq_{\mathsf{B}} c$ *for every valid deduction.*

Proof. By reverse application of the final rule, $\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{b'}; \mathbb{C}_1$ where $b' \leq_{\mathsf{B}} b$ by Proposition 4.3.9. Then by Theorem 4.3.13, $\mathbb{R}(\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P}' : \mathsf{Proc}^{b'}; \mathbb{C}'_1)$ where $\mathbb{C}' \subseteq \mathbb{C}$, and hence by the final rule

$$\mathbb{R}(\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P}' : \mathsf{Proc}^{c})$$

where $c = \bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}'_{\mathrm{T}}(x) \geq_{\mathsf{B}} b$ by Proposition 4.3.9 as required. □

4.3.4 Security Properties

A security property for system $\mathcal{T}_{\mathsf{HO}\pi\leq}$ effectively provides the *semantics* for the type system. That is, it establishes exactly what is guaranteed by the final annotation on a program.

The key idea is the preservation of the intuition introduced at the beginning; that a collection of processes be considered untrusted in total if any of the sub-processes contained within are both untrusted and capable of communication with the host, through the pre-defined set of channels specified by $\mathbb{C}_\epsilon$. The same should apply if this situation might develop at any point during a reduction. So, the security property states that a process considered trusted may never evolve such that an untrusted sub process occurs guarded by one of the names in $\mathbb{C}_\epsilon$.

This is accomplished in two steps; firstly a lemma stating that any sub-process guarded by a name in $\mathbb{C}_\epsilon$ must have a trustedness at least as safe as the trustedness of the whole process, and secondly a theorem building on this result and subject reduction to guarantee that this will always be the case for any reduction.

Two simple pieces of notation are needed first. Both definitions are due to Milner (1993).

Definition 4.3.15 (Unguardedness) *A process* P *is said to occur* unguarded *in* Q *if it is a sub-term of* Q *and not under a guard. Inductively:*

- P *is unguarded in* P
- *If* P *is unguarded in* Q *then* P *is also unguarded in* !Q
- *If* P *is unguarded in* Q *then* P *is also unguarded in* Q|R

- *If* P *is unguarded in* Q *then* P *is also unguarded in* $\mathrm{Q} + \mathrm{R}$

Definition 4.3.16 (Observability) *A process* P *is observable at* x*, written* $\mathrm{P}{\downarrow_x}$*, if there is some process* $\mu.\mathrm{Q}$ *such that* $\mu.\mathrm{Q}$ *occurs unguarded in* P*, and* x *is the subject of the guard* μ*. Inductively:*

- $(x(\overrightarrow{V} : \overrightarrow{\sigma^b}).\mathrm{P}){\downarrow_x}$
- $(\overline{x}[\overrightarrow{V}].\mathrm{P}){\downarrow_x}$
- $(x(V : \sigma^i)_{\mathrm{certify}}^{?}\mathrm{P} \oplus \mathrm{Q}){\downarrow_x}$
- $\mathrm{P}{\downarrow_x}$ *implies:*
  - $(!\mathrm{P}){\downarrow_x}$
  - $(\mathrm{P}|\mathrm{Q}){\downarrow_x}$
  - $(\mathrm{P} + \mathrm{Q}){\downarrow_x}$
  - $((\nu y : \sigma^b)\mathrm{P}){\downarrow_x}$*, if* $y \neq x$

The inverse of the relation, "not observable at", is written $\mathrm{P}{\not\downarrow_x}$*. A weaker form,* $\mathrm{P}{\Downarrow_x}$ *is defined if there is some* P' *such that* P *reduces to* P'*, and* $\mathrm{P}'{\downarrow_x}$*.*

Now the main lemma can be stated using these definitions; note that it relies on the deduction being deterministic (that is, modulo order of composition and summation, and the use of weakening, there is a single derivation for a given term and environment).

Lemma 4.3.17 $\forall \Gamma, \mathrm{P}, b$*. If there is a valid deduction in System* $\mathcal{T}_{\mathsf{HO}\pi\leq}$ *for*

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b$$

then for all $x \in \mathbb{C}_\epsilon$ *if* $\mathrm{P}{\downarrow_x}$ *then for every* Q *such that* $\mu.\mathrm{Q}$ *occurs unguarded in* P *and* x *is the subject of* μ*, there is a deduction*

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mu.\mathrm{Q} : \mathsf{Proc}^c; \mathbb{C}$$

where $c \geq_{\mathsf{B}} b$*.*

Proof. By cases on the definition of observability and then by induction on the derivation of $\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b$. By Definition 4.3.16 there are three base cases to account for, and four structural cases. The base cases of input, output, and certify are trivial by the statement, and the structural cases follow by the appropriate type rules and the induction hypothesis.

- $(x(\overrightarrow{V} : \overrightarrow{\sigma^c}).\mathrm{P})\downarrow_x$: By the statement there is a derivation for

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} x(\overrightarrow{V} : \overrightarrow{\sigma^c}).\mathrm{P} : \mathsf{Proc}^b$$

This must have ended in a use of the Final rule, with antecedent

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} x(\overrightarrow{V} : \overrightarrow{\sigma^c}).\mathrm{P} : \mathsf{Proc}^{b'}\mathbb{C}_{\overrightarrow{V}}, x : b'$$

where $b' \leq_{\mathsf{B}} b$ by Lemma 4.3.9. However, noting that since $x \in \mathbb{C}_\epsilon$ and the context of x is b', it must be the case that $b = b'$, and hence result holds for in this case.

- $(\overline{x}[\overrightarrow{V}].\mathrm{P})\downarrow_x$: Similarly.

- $(x(V : \sigma^i)_{\mathrm{certify}}\mathrm{P} \oplus \mathrm{Q})\downarrow_x$: Similarly.

- $\mathrm{P}\downarrow_x$ implies $(!\mathrm{P})\downarrow_x$, and $x \in \mathbb{C}_\epsilon$: By the induction hypothesis.

- $\mathrm{P}\downarrow_x$ implies $((\nu y : \sigma^b)\mathrm{P})\downarrow_x$, provided $y \neq x$, and $x \in \mathbb{C}_\epsilon$: By the induction hypothesis.

- $\mathrm{P}\downarrow_x$ implies $(\mathrm{P}|\mathrm{Q})\downarrow_x$, and $x \in \mathbb{C}_\epsilon$: By the statement, there is a deduction for

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P}|\mathrm{Q} : \mathsf{Proc}^b$$

This deduction must have ended in a use of the Final rule, with antecedent

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P}|\mathrm{Q} : \mathsf{Proc}^{b'}; \mathbb{C}$$

for some $\mathbb{C}$, where $b' \leq_{\mathsf{B}} b$ by Lemma 4.3.9. This deduction itself must have ended in zero or more uses of the weaken rule, preceded by an instance of the composition rule with antecedents

$$\Gamma' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{b'_1}; \mathbb{C}_1$$

and

$$\Gamma'' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{Q} : \mathsf{Proc}^{b'_2}; \mathbb{C}_2$$

where $\Gamma' \cup \Gamma'' \subseteq \Gamma$ by the weaken rule, $\mathbb{C} = \mathbb{C}_1 \cup \mathbb{C}_2$, and $b' = b'_1 \cdot b'_2$. By the induction hypothesis the result holds for P; that is, applying the Final rule to this deduction produces

$$\Gamma' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{b_1}$$

where $b_1 = \bigwedge_{x \in \mathbb{C}_\epsilon} \mathbb{C}_{1\mathrm{T}}(x)$ and $b_1 \geq_{\mathsf{B}} b_1'$ by Lemma 4.3.9. Then for all $x \in \mathbb{C}_\epsilon$, for all R such that μ.R occurs unguarded in P where x is the subject of μ, there is a deduction

$$\Gamma' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{R} : \mathsf{Proc}^d; \mathbb{C}_R$$

for some $d \geq_{\mathsf{B}} b_2$. Hence, noting by transitivity of $\leq_{\mathsf{B}}$ that $d \geq_{\mathsf{B}} b_2 \geq_{\mathsf{B}} b$, result holds for P|Q as well.

- $\mathrm{P}{\downarrow_x}$ implies $(\mathrm{P} + \mathrm{Q}){\downarrow_x}$, and $x \in \mathbb{C}_\epsilon$: similarly.

□

The main security theorem is an extension of Lemma 4.3.17, and states that the same result holds after an arbitrary number of reductions: that is, if a process is judged to be trusted (with respect to that host) then it will never evolve such that an untrusted process may communicate with the host.

Theorem 4.3.18 $\forall \Gamma, \mathrm{P}, b$. *If there is a valid deduction in System* $\mathcal{T}_{\mathsf{HO}\pi\leq}$ *for*

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b$$

then for every P', $\mathbb{C}$, $\mathbb{C}'$, *and* $\mathbb{R}$ *such that (for every* Θ, $\mathbb{C}''$*)*

$$\langle \Theta | \mathbb{C}'', \Gamma | \mathbb{C}, \mathrm{P} \rangle \overset{\vec{\tau}}{\twoheadrightarrow} \langle \Theta | \mathbb{C}'', \Gamma' | \mathbb{C}', \mathrm{P}' \rangle \Vdash_{\mathrm{certify}} \mathbb{R}$$

then for all $x \in \mathbb{C}_\epsilon$ *if* $\mathrm{P}'{\downarrow_x}$ *(that is,* $\mathrm{P}{\Downarrow_x}$*) then for every* Q *such that* μ.Q *occurs unguarded in* P *and* x *is the subject of* μ*, there is a deduction*

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mu.\mathrm{Q} : \mathsf{Proc}^c; \mathbb{C}_Q$$

where $c \geq_{\mathsf{B}} b$.

Proof. By subject reduction (Theorem 4.3.14) and Lemma 4.3.17. □

4.4 Discussion

This Chapter demonstrated the safety of the systems of the previous chapter from two perspectives: a proof of subject reduction, and security. Only three of the four systems; Systems $\mathcal{T}_{\mathsf{FO}\pi}$, $\mathcal{T}_{\mathsf{FO}'\pi\leq}$, and $\mathcal{T}_{\mathsf{HO}\pi\leq}$, were examined, since System $\mathcal{T}_{\mathsf{FO}\pi\leq}$ was presented mainly as an intermediate step in the development of System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$.

Subject reduction is a guarantee that types are preserved under reduction, so the user may be assured that whatever properties they have enforced

via the type system in their original program will continue to hold as the program executes. The proof consists of a preliminary substitution lemma to guarantee that a name or sub-term may be replaced by another of a similar type without affecting the type of the overall term, then the subject reduction result itself, which relies on the substitution lemma during the base cases of communication reduction.

The situation is complicated in the systems of this thesis due to the possible presence of run-time coercion during a reduction. This is factored in by including the annotation substitution, generated by the typed labelled transition semantics, in the consequent of the statement. (That is, the deduction is the same before and after reduction, modulo the substitutions on type annotations generated by the reduction).

Security results were presented for Systems $\mathcal{T}_{\mathsf{FO}'\pi\leq}$ and $\mathcal{T}_{\mathsf{HO}\pi\leq}$ only, as System $\mathcal{T}_{\mathsf{FO}\pi}$ is not a security-based system (other than that provided by a guarantee of Subject Reduction). The Security Property of System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$ was established as an instance of Strong Non-deterministic Non-Interference: a guarantee that the execution of a process typed at a certain security level will not be affected by channels of a lower integrity. This was established by examining the traces generated by the typed labelled transition semantics, and determining that the traces obtained by removing all names of a lower integrity than the process' security level were also valid traces.

Security for System $\mathcal{T}_{\mathsf{HO}\pi\leq}$ instead focuses on observability of processes, from a predetermined perspective (the external context $\mathbb{C}_\epsilon$). A statement is proven that if a process is typed as trusted after the rule Final then it will never reduce such an untrusted process is observable at any of the names in the external context. This allows a collection of processes of different levels of trustedness to be typed at a level that reflects their ability to communicate with the host, so a collection containing untrusted processes may still be typed as trusted, providing none of the untrusted components are ever in a position to communicate with the host.

Chapter 5

Implementation

5.1 Introduction

This chapter describes type inference[1] algorithms for the three systems presented in Chapter 3, and demonstrates their correctness by showing that they are all both sound and complete, and always terminate.

5.2 Preliminaries

The basic mode of operation of all the inference algorithms in this chapter is to assign new arbitrary types to names and variables when encountered, then when two branches are merged to reconcile (make equivalent) the types assigned to the names and variables present in both branches.

This is accomplished by introducing type variables (Definition 5.2.1), so that each new name encountered can be assigned a fresh type variable. Then, appropriate algorithms return substitutions (Definition 5.2.2) that equate the types.

Definition 5.2.1 (Type Variables) *Extend the syntax of types with type variables, ranged over by α. Thus, α^i is a variable base type with a variable annotation. The set of all type variables is represented by $\mathcal{T}_\mathcal{V}$, where $\mathcal{T}_\mathcal{V} \subset \mathcal{T}$.*

[1] Some presentations refer to verifying the well-formedness — and possibly assigning a type, if appropriate — of an explicitly-typed term as 'type inference'. Here the term 'inference' is used to refer to the problem of assigning types to all free variables in a term (with no initial assumptions) such that the term is well typed. This task is sometimes referred to as "type reconstruction"; see for example Pierce (2002). The difference is actually less clear-cut here, as bound names are explicitly typed.

Definition 5.2.2 (Substitutions) *(This largely follows Wright (1992, Definition 5.1.2)) A substitution is a pair*

$$(\mathbb{S}_T \in \mathcal{T}_\mathcal{V} \to \mathcal{T}, \mathbb{R}_B \in \mathcal{B}_\mathcal{V} \to \mathcal{B}_\mathcal{A})$$

A few conventions and notational shorthands will be employed throughout the remainder of this thesis (let $\mathbb{S} = (\mathbb{S}_T, \mathbb{R}_B)$ *in the following):*

1. *Substitutions will usually be written just as* $\mathbb{S}$*; the substitution that operates* only *on annotations will be written* $\mathbb{R}$ *as before, to denote* $(\mathsf{Id}, \mathbb{R}_B)$*.*

2. *The notation* Id *will denote* $(\mathsf{Id}, \mathsf{Id})$*.*

3. *When combining substitutions, the preference is to use the in-order composition* $\mathbb{S}_1; \mathbb{S}_2$ *(that is, apply* $\mathbb{S}_1$*, then* $\mathbb{S}_2$*) in place of the more common* $\mathbb{S}_2 \circ \mathbb{S}_1$*.*

4. *Application:*

   - $\mathbb{S}(\sigma^b) = \mathbb{S}_T; \mathbb{R}_B(\sigma^b)$
   - $\mathbb{S}((\sigma_1^b \dots \sigma_n^c)^d) = (\mathbb{S}(\sigma_1^b) \dots \mathbb{S}(\sigma_n^c))^{\mathbb{S}(d)}$

5. *Write* $\mathbb{S}[\alpha^i := \sigma^b]$ *for* $(\mathbb{S}_T[\alpha := \sigma], \mathbb{R}_B[i := b])$ *and* $\mathbb{S}[i := b]$ *for* $(\mathbb{S}_T, \mathbb{R}_B[i := b])$*. Similarly, write* $\mathbb{S}; \mathbb{R}$ *for* $(\mathbb{S}_T, \mathbb{R}_B; \mathbb{R})$*.*

6. *Define the ordering* $\leq$ *as* $\mathbb{S} \leq \mathbb{S}' \iff \exists \mathbb{S}''. \mathbb{S}; \mathbb{S}'' = \mathbb{S}'$*.*

Definition 5.2.3 *Write* $\Gamma \leq \Gamma'$ *if and only if there is some substitution* $\mathbb{S}$ *such that* $\mathbb{S}(\Gamma) \subseteq \Gamma'$*.*

Definition 5.2.4 *Write* $\mathbb{C} \leq \mathbb{C}'$ *if and only if there is some substitution* $\mathbb{R}$ *such that* $\mathbb{R}(\mathbb{C}) \subseteq \mathbb{C}'$*.*

5.2.1 Unification

A fundamental operation in most inference algorithms is that of *unification*. Intuitively, the reason for this is easy to see: inference algorithms generally proceed recursively along all branches of the deduction tree, assigning fresh variables as types for all names not previously encountered. When two or more branches of a deduction must be combined (for example, using the composition rule to combine two processes) all this accumulated information about the types assigned to names must be reconciled (so that a name has the same type in both branches, that they may correctly be combined); this reconciliation is achieved via unification.

Unification then is the process of deriving a substitution such that the image of any pair of types thus unified under the substitution is identical. Several algorithms for achieving this, corresponding to the different situations in which substitution is required, are presented here before proceeding with the inference algorithms themselves commencing in Section 5.3. The unification algorithms are displayed first, followed by proofs of their correctness (soundness and completeness) in Section 5.2.2

Definition 5.2.5 *The algorithm* BUNIFY *returns the most general substitution (if one exists) unifying its two annotation arguments. It is simply the boolean unification algorithm from, for example, Boole (1847); although any such algorithm would suffice. It is sound and complete; see Martin and Nipkow (1990). It terminates for all finite inputs.*

Definition 5.2.6 *The algorithm* $\mathcal{U}$ *returns the most general substitution (if one exists) unifying its two type arguments. To avoid repeating the algorithm, process types (that is, of the form* $(\overrightarrow{\sigma^b}) \to \mathsf{Proc}^c$*) are included, as the algorithm is otherwise unchanged for the higher-order system. It is sound and complete; see Theorems 5.2.10 and 5.2.11 respectively.*

$$
\begin{aligned}
\mathcal{U}\left((\overrightarrow{\sigma_1^{b'}})^b, (\overrightarrow{\sigma_2^{c'}})^c\right) &= \mathit{let}\ \mathbb{R} = \mathsf{BUNIFY}(b,c) \\
&\quad \mathit{in}\ \mathbb{R}; \mathcal{U}\left(\mathbb{R}(\overrightarrow{\sigma_1^{b'}}), \mathbb{R}(\overrightarrow{\sigma_2^{c'}})\right) \\
\mathcal{U}\big(\mathsf{Proc}^b, \mathsf{Proc}^c\big) &= \mathsf{BUNIFY}(b,c) \\
\mathcal{U}\left((\overrightarrow{\sigma_1^{b'}}) \to \mathsf{Proc}^b, (\overrightarrow{\sigma_2^{c'}}) \to \mathsf{Proc}^c\right) &= \mathit{let}\ \mathbb{R} = \mathsf{BUNIFY}(b,c) \\
&\quad \mathit{in}\ \mathbb{R}; \mathcal{U}\left(\mathbb{R}(\overrightarrow{\sigma_1^{b'}}), \mathbb{R}(\overrightarrow{\sigma_2^{c'}})\right) \\
\mathcal{U}\big(\sigma^b, \alpha^c\big) &= \mathcal{U}\big(\alpha^c, \sigma^b\big) \\
\mathcal{U}\big(\alpha^c, \sigma^b\big) &= \mathit{let}\ \mathbb{R} = \mathsf{BUNIFY}(b,c) \\
&\quad \mathit{in}\ \mathbb{R}; \mathsf{Id}[\mathbb{R}(\alpha) := \mathbb{R}(\sigma)] \\
\mathcal{U}\left(\sigma_1^b \overrightarrow{\sigma_1'^{b'}}, \sigma_2^c \overrightarrow{\sigma_2'^{c'}}\right) &= \mathit{if}\ \mathit{length}\left(\overrightarrow{\sigma_1'^{b'}}\right) \neq \mathit{length}\left(\overrightarrow{\sigma_2'^{c'}}\right)\ \mathit{then}\ \bot \\
&\quad \mathit{else}\ \mathit{let}\ \mathbb{S} = \mathcal{U}\big(\sigma_1^b, \sigma_2^c\big) \\
&\quad \mathit{in}\ \mathbb{S}; \mathcal{U}\left(\mathbb{S}(\overrightarrow{\sigma_2'^{c'}}), \mathbb{S}(\overrightarrow{\sigma_2'^{c'}})\right)
\end{aligned}
$$

Definition 5.2.7 *The algorithm* Unify *is used to unify environments. It returns the most general substitution, if one exists, that unifies the images (which are types) of all names in the domain of both its arguments.*

$$\begin{aligned} \mathsf{Unify}(\Gamma, \Theta) &= \mathsf{Id} \quad (\mathrm{dom}\Gamma \cap \mathrm{dom}\Theta = \emptyset) \\ \mathsf{Unify}\big(\{x : \sigma_1^b\} \cup \Gamma, \{x : \sigma_2^c\} \cup \Theta\big) &= \mathbb{S}; \mathsf{Unify}(\mathbb{S}(\Gamma), \mathbb{S}(\Theta)) \\ &\quad \textit{where } \mathbb{S} = \mathcal{U}\big(\sigma_1^b, \sigma_2^c\big) \end{aligned}$$

It is sound and complete; see Theorems 5.2.13 and 5.2.14 respectively.

Definition 5.2.8 *The algorithm* CUnify *is used to unify contexts. It creates a substitution unifying the images, which are annotations, of all names appearing in the domain of both its arguments.*

$$\begin{aligned} \mathsf{CUnify}(\mathbb{C}_1, \mathbb{C}_2) &= \mathsf{Id} \quad (\mathrm{dom}\mathbb{C}_1 \cap \mathrm{dom}\mathbb{C}_2 = \emptyset) \\ \mathsf{CUnify}(\{x : b\} \cup \mathbb{C}_1, \{x : c\} \cup \mathbb{C}_2) &= \mathbb{R}; \mathsf{CUnify}(\mathbb{R}(\mathbb{C}_1), \mathbb{R}(\mathbb{C}_2)) \\ &\quad \textit{where } \mathbb{R} = \mathsf{BUNIFY}(b, c) \end{aligned}$$

It is sound and complete; see Theorems 5.2.16 and 5.2.17 respectively.

Definition 5.2.9 *The algorithm* $\mathbb{M}$ *returns a substitution such that, if* $\mathbb{R} = \mathbb{M}(b, \sigma^c)$*, then* $\mathbb{R}(c) = \mathbb{R}(b \cdot c)$*. This is used to ensure the well-formedness of channel types, recalling that for all systems the carried type annotations must be products of the outermost annotation. The variable* i *in the following is a fresh annotation variable.*

$$\begin{aligned} \mathbb{M}\big(b, \sigma^{\mathrm{T}}\big) &= \mathsf{BUNIFY}(b, \mathrm{T}) \\ \mathbb{M}\big(b, \sigma^{\mathrm{U}}\big) &= \mathsf{Id} \\ \mathbb{M}(b, \sigma^c) &= \mathsf{BUNIFY}(c, b \cdot i) \\ \mathbb{M}\Big(b, \sigma^c \overrightarrow{\sigma'^e}\Big) &= \textit{let } \mathbb{S} = \mathbb{M}(b, \sigma^c) \\ &\quad \textit{in } \mathbb{S}; \mathbb{M}\Big(\mathbb{S}(b), \mathbb{S}(\overrightarrow{\sigma'^d})\Big) \end{aligned}$$

Note that if the type annotation is T *then the "multiplying" annotation* b *must be equivalent (that is, unified) to* T *since* $b \cdot \mathrm{T} = b$*. Alternatively, since* $\forall b. b \cdot \mathrm{U} = \mathrm{U}$ *then if the annotation is* U *the identity substitution suffices. For all other cases, the annotation is unified with the multiplication of a fresh variable and* b*.*

It is sound and complete; see Theorems 5.2.19 and 5.2.20 respectively.

5.2.2 Correctness of the Unification Procedures

Theorem 5.2.10 (Soundness of $\mathcal{U}$)

$$\forall \sigma_1^b, \sigma_2^c.\mathbb{S} = \mathcal{U}\big(\sigma_1^b, \sigma_2^c\big) \Longrightarrow \mathbb{S}(\sigma_1^b) = \mathbb{S}(\sigma_2^c)$$

Proof. By cases on the structure of the arguments.

- $\mathcal{U}\Big((\overrightarrow{\sigma_1^{b'}})^b, (\overrightarrow{\sigma_2^{c'}})^c\Big)$: From the algorithm definition and the soundness of BUNIFY, $\mathbb{R} = \mathsf{BUNIFY}(b, c) \Longrightarrow \mathbb{R}(b) = \mathbb{R}(c)$. By the induction hypothesis $\mathbb{S} = \mathbb{R}; \mathcal{U}\Big(\mathbb{R}(\overrightarrow{\sigma_1^{b'}}), \mathbb{R}(\overrightarrow{\sigma_2^{c'}})\Big) \Longrightarrow \mathbb{S}(\overrightarrow{\sigma_1^{b'}}) = \mathbb{S}(\overrightarrow{\sigma_2^{c'}})$, hence soundness holds in this case too.
- $\mathcal{U}\Big((\overrightarrow{\sigma_1^{b'}}) \to \mathsf{Proc}^b, (\overrightarrow{\sigma_2^{c'}}) \to \mathsf{Proc}^c\Big)$: Similarly.
- $\mathcal{U}\big(\mathsf{Proc}^b, \mathsf{Proc}^c\big)$: By soundness of BUNIFY.
- $\mathcal{U}\big(\alpha^b, \sigma^c\big)$: By the algorithm this is defined as $\mathbb{R}; \mathsf{Id}[\mathbb{R}(\alpha) := \mathbb{R}(\sigma)]$ (where $\mathbb{R} = \mathsf{BUNIFY}(b, c)$), so by inspection if $\mathbb{S} = \mathcal{U}\big(\alpha^b, \sigma^c\big)$ then $\mathbb{S}(\alpha^b) = \mathbb{R}(\sigma^c)$ and $\mathbb{S}(\sigma^c) = \mathbb{R}(\sigma^c)$ so soundness holds.
- $\mathcal{U}\big(\sigma^b, \alpha^c\big)$: By definition of the algorithm this is defined as $\mathcal{U}\big(\alpha^c, \sigma^b\big)$, and as shown above this is sound.
- $\mathcal{U}\left(\sigma_1^b \overrightarrow{\sigma_1'^{b'}}, \sigma_2^c \overrightarrow{\sigma_2'^{c'}}\right)$: By the statement $\mathit{length}\left(\overrightarrow{\sigma_1'^{b'}}\right) = \mathit{length}\left(\overrightarrow{\sigma_2'^{c'}}\right)$ so by inspection of the algorithm this case is defined as $\mathbb{S}; \mathcal{U}\left(\mathbb{S}(\overrightarrow{\sigma_2'^{c'}}), \mathbb{S}(\overrightarrow{\sigma_2'^{c'}})\right)$ where $\mathbb{S} = \mathcal{U}\big(\sigma_1^b, \sigma_2^c\big)$. By the induction hypothesis $\mathbb{S}$ is sound, and by the first case above $\mathbb{S}; \mathcal{U}\left(\mathbb{S}(\overrightarrow{\sigma_2'^{c'}}), \mathbb{S}(\overrightarrow{\sigma_2'^{c'}})\right)$ is also sound, as required.

□

Theorem 5.2.11 (Completeness of $\mathcal{U}$) $\forall \sigma_1^b, \sigma_2^c$. *If* $\exists \mathbb{S}'.\mathbb{S}'(\sigma_1^b) = \mathbb{S}'(\sigma_2^c)$ *then* $\mathbb{S} = \mathcal{U}\big(\sigma_1^b, \sigma_2^c\big)$ *is defined, and* $\mathbb{S} \leq \mathbb{S}'$.

Proof. By inspection of the algorithm and cases on the structure of σ_1^b, σ_2^c.

- $\mathcal{U}\Big((\overrightarrow{\sigma_1^{b'}})^b, (\overrightarrow{\sigma_2^{c'}})^c\Big)$: From the algorithm definition this case is defined as $\mathbb{S} = \mathbb{R}; \mathcal{U}\Big(\mathbb{R}(\overrightarrow{\sigma_1^{b'}}), \mathbb{R}(\overrightarrow{\sigma_2^{c'}})\Big)$ where $\mathbb{R} = \mathsf{BUNIFY}(b, c)$. By completeness of BUNIFY, $\mathbb{R}$ is defined and $\mathbb{R} \leq \mathbb{S}'$. By the induction hypothesis

$\mathcal{U}\left(\mathbb{R}(\overrightarrow{\sigma_1^{b'}}), \mathbb{R}(\overrightarrow{\sigma_2^{c'}})\right) \leq \mathbb{S}'$ (also implying $\mathcal{U}$ is defined in this instance), hence completeness holds in this case too. Note that this assumes the lengths of $\overrightarrow{\sigma_1^{b'}}$ and $\overrightarrow{\sigma_2^{c'}}$ are identical; according to the statement $\mathbb{S}'$ is defined so this must be the case.

- $\mathcal{U}\left((\overrightarrow{\sigma_1^{b'}}) \to \mathsf{Proc}^b, (\overrightarrow{\sigma_2^{c'}}) \to \mathsf{Proc}^c\right)$: Similarly.
- $\mathcal{U}\left(\mathsf{Proc}^b, \mathsf{Proc}^c\right)$: By completeness of BUNIFY.
- $\mathcal{U}\left(\alpha^b, \sigma^c\right)$: By definition of the algorithm this is defined — given $\mathbb{R} = \mathsf{BUNIFY}(b, c)$ — as $\mathbb{R}; \mathsf{Id}[\mathbb{R}(\alpha^b) := \mathbb{R}(\sigma^c)]$, so by inspection $\mathbb{S}$ is defined and $\mathbb{S} \leq \mathbb{S}'$ so completeness holds.
- $\mathcal{U}\left(\sigma^b, \alpha^c\right)$: By definition of the algorithm this is defined as $\mathcal{U}\left(\alpha^c, \sigma^b\right)$, and as shown above completeness holds for this case.
- $\mathcal{U}\left(\sigma_1^b \overrightarrow{\sigma_1'^{b'}}, \sigma_2^c \overrightarrow{\sigma_2'^{c'}}\right)$: By the statement since $\mathbb{S}'$ is defined then it must be the case that $\mathit{length}\left(\overrightarrow{\sigma_1'^{b'}}\right) = \mathit{length}\left(\overrightarrow{\sigma_2'^{c'}}\right)$. By inspection the algorithm is defined as $\mathbb{S}; \mathcal{U}\left(\mathbb{S}(\overrightarrow{\sigma_2'^{c'}}), \mathbb{S}(\overrightarrow{\sigma_2'^{c'}})\right)$ where $\mathbb{S} = \mathcal{U}\left(\sigma_1^b, \sigma_2^c\right)$. Hence by the induction hypothesis completeness holds for $\mathbb{S}$, and by the first case above $\mathbb{S}; \mathcal{U}\left(\mathbb{S}(\overrightarrow{\sigma_2'^{c'}}), \mathbb{S}(\overrightarrow{\sigma_2'^{c'}})\right)$ is also complete, as required.

□

Theorem 5.2.12 (Termination of $\mathcal{U}$) *The algorithm $\mathcal{U}$ terminates for all finite inputs.*

Proof. By induction on the inputs. Each step either terminates (for example, with a call to BUNIFY which also terminates), or results in a recursive call on a subset of the inputs. As each step operates on a successively smaller input, the algorithm eventually terminates.

For example, consider $\mathcal{U}\left(\sigma_1^b \overrightarrow{\sigma_1'^{b'}}, \sigma_2^c \overrightarrow{\sigma_2'^{c'}}\right)$: by examination of the algorithm, if the lengths of the tail sequences are not equal the algorithm terminates. Otherwise, there are two recursive calls; by the induction hypothesis $\mathbb{S} = \mathcal{U}\left(\sigma_1^b, \sigma_2^c\right)$ terminates, and also by the induction hypothesis the recursive call $\mathcal{U}\left(\mathbb{S}(\overrightarrow{\sigma_2'^{c'}}), \mathbb{S}(\overrightarrow{\sigma_2'^{c'}})\right)$ also terminates. □

Theorem 5.2.13 (Soundness of Unify)

$$\forall \Gamma, \Theta. \mathbb{S} = \mathsf{Unify}(\Gamma, \Theta) \Longrightarrow \mathbb{S}(\Gamma) \asymp \mathbb{S}(\Theta)$$

Proof. By induction on the length of the intersection $\mathrm{dom}\Gamma \cap \mathrm{dom}\Theta$.

- If $\mathrm{dom}\Gamma \cap \mathrm{dom}\Theta = \emptyset$ then $\mathbb{S} = \mathsf{Unify}(\Gamma, \Theta) = \mathsf{Id}$, and $\mathbb{S}(\Gamma) \asymp \mathbb{S}(\Theta) = \Gamma \asymp \Theta$ is trivially defined (see Definition 2.3.1 on page 33 for the definition of $\asymp$).
- If $\exists x.\Gamma = \Gamma', x : \sigma_1^b$ and $\Theta = \Theta', x : \sigma_2^c$ then by inspection of the algorithm $\mathbb{S} = \mathbb{S}'; \mathsf{Unify}(\mathbb{S}'(\Gamma'), \mathbb{S}'(\Theta'))$ where $\mathbb{S}' = \mathcal{U}\left(\sigma_1^b, \sigma_2^c\right)$. By soundness of $\mathcal{U}$ (Theorem 5.2.10, above) $\mathbb{S}'(\sigma_1^b) = \mathbb{S}'(\sigma_2^c)$, and by the induction hypothesis if $\mathbb{S}'' = \mathsf{Unify}(\mathbb{S}'(\Gamma'), \mathbb{S}'(\Theta'))$ is defined then $\mathbb{S}''(\mathbb{S}'(\Gamma')) \asymp \mathbb{S}''(\mathbb{S}'(\Theta'))$ is defined. Hence $\mathbb{S}(\Gamma) \asymp \mathbb{S}(\Theta)$ as required, where $\mathbb{S} = \mathbb{S}'; \mathbb{S}''$.

□

Theorem 5.2.14 (Completeness of Unify) $\forall \Gamma, \Theta$. *If* $\exists \mathbb{S}'.\mathbb{S}'(\Gamma) \asymp \mathbb{S}'(\Theta)$ *then* $\mathbb{S} = \mathsf{Unify}(\Gamma, \Theta)$ *is defined, and* $\mathbb{S} \leq \mathbb{S}'$.

Proof. By induction on the length of the intersection $\mathrm{dom}\Gamma \cap \mathrm{dom}\Theta$.

- If $\mathrm{dom}\Gamma \cap \mathrm{dom}\Theta = \emptyset$ then $\mathbb{S} = \mathsf{Unify}(\Gamma, \Theta) = \mathsf{Id}$ (by definition of the algorithm), and $\mathsf{Id} \leq \mathbb{S}'$ for all $\mathbb{S}'$.
- If $\exists x.\Gamma = \Gamma_1', x : \sigma_1^b$ and $\Theta = \Theta', x : \sigma_2^c$ then by inspection of the algorithm $\mathbb{S} = \mathbb{S}''; \mathsf{Unify}(\mathbb{S}''(\Gamma'), \mathbb{S}'(\Theta'))$ where $\mathbb{S}'' = \mathcal{U}\left(\sigma_1^b, \sigma_2^c\right)$. By completeness of $\mathcal{U}$ (Theorem 5.2.11, above) $\mathbb{S}''$ is defined and most general (i.e. $\mathbb{S}'' \leq \mathbb{S}'$), and by the induction hypothesis $\mathsf{Unify}(\mathbb{S}''(\Gamma'), \mathbb{S}''(\Theta'))$ is also defined and produces a most general unifier. Then $\mathbb{S} = \mathsf{Unify}(\Gamma, \Theta)$ is defined, and $\mathbb{S} \leq \mathbb{S}'$ as required.

□

Theorem 5.2.15 (Termination of Unify) *The algorithm* Unify *terminates for every finite set of inputs.*

Proof. By induction on the length of intersection of the domain of the inputs. There are two cases to consider; if the intersection is empty then the algorithm terminates. If there is at least one pair in the intersection, the algorithm calls $\mathcal{U}$ which terminates by Theorem 5.2.12, then makes a recursive call on the remainder of the inputs. Since each step either terminates or reduces the size of the inputs, the algorithm eventually terminates.

For example, consider the case of $\mathsf{Unify}\big(\{x : \sigma_1^b\} \cup \Gamma, \{x : \sigma_2^c\} \cup \Theta\big)$: There are two steps involved; by Theorem 5.2.12 $\mathbb{S} = \mathcal{U}\big(\sigma_1^b, \sigma_2^c\big)$ terminates, and hence by the induction hypothesis the recursive call on the remainder of the input $\mathsf{Unify}(\mathbb{S}(\Gamma), \mathbb{S}(\Theta))$ also terminates. □

Theorem 5.2.16 (Soundness of CUnify)

$$\forall \mathbb{C}_1, \mathbb{C}_2.\mathbb{R} = \mathsf{CUnify}(\mathbb{C}_1, \mathbb{C}_2) \Longrightarrow \mathbb{R}(\mathbb{C}_1) \asymp \mathbb{R}(\mathbb{C}_2)$$

Proof. By induction on the length of the intersection $\mathrm{dom}\mathbb{C}_1 \cap \mathrm{dom}\mathbb{C}_2$.

- If $\mathrm{dom}\mathbb{C}_1 \cap \mathrm{dom}\mathbb{C}_2 = \emptyset$ then $\mathbb{R} = \mathsf{CUnify}(\mathbb{C}_1, \mathbb{C}_2) = \mathsf{Id}$, and $\mathbb{R}(\mathbb{C}_1) \asymp \mathbb{R}(\mathbb{C}_2) = \mathbb{C}_1 \asymp \mathbb{C}_2$ is trivially defined (see page 67 for the definition of $\asymp$).

- If $\exists x.\mathbb{C}_1 = \mathbb{C}_1', x : b$ and $\mathbb{C}_2 = \mathbb{C}_2', x : c$ then by inspection of the algorithm $\mathbb{R} = \mathbb{R}'; \mathsf{CUnify}(\mathbb{R}'(\mathbb{C}_1'), \mathbb{R}'(\mathbb{C}_2'))$ where $\mathbb{R}' = \mathsf{BUNIFY}(b, c)$. By soundness of BUNIFY $\mathbb{R}'(b) = \mathbb{R}'(c)$, and by the induction hypothesis if $\mathbb{R}'' = \mathsf{CUnify}(\mathbb{R}'(\mathbb{C}_1'), \mathbb{R}'(\mathbb{C}_2'))$ is defined then $\mathbb{R}''(\mathbb{R}'(\mathbb{C}_1')) \asymp \mathbb{R}''(\mathbb{R}'(\mathbb{C}_2'))$ is defined. Hence $\mathbb{R}(\mathbb{C}_1) \asymp \mathbb{R}(\mathbb{C}_2)$ is defined as required, where $\mathbb{R} = \mathbb{R}''; \mathbb{R}'$.

□

Theorem 5.2.17 (Completeness of CUnify) $\forall \mathbb{C}_1, \mathbb{C}_2$. *If* $\exists \mathbb{R}'.\mathbb{R}'(\mathbb{C}_1) \asymp \mathbb{R}'(\mathbb{C}_2)$ *then* $\mathbb{R} = \mathsf{CUnify}(\mathbb{C}_1, \mathbb{C}_2)$ *is defined, and* $\mathbb{R} \leq \mathbb{R}'$.

Proof. By induction on the length of the intersection $\mathrm{dom}\mathbb{C}_1 \cap \mathrm{dom}\mathbb{C}_2$.

- If $\mathrm{dom}\mathbb{C}_1 \cap \mathrm{dom}\mathbb{C}_2 = \emptyset$ then $\mathbb{R} = \mathsf{CUnify}(\mathbb{C}_1, \mathbb{C}_2) = \mathsf{Id}$, and $\mathsf{Id} \leq \mathbb{R}'$ for all $\mathbb{R}'$.

- If $\exists x.\mathbb{C}_1 = \mathbb{C}_1', x : b$ and $\mathbb{C}_2 = \mathbb{C}_2', x : c$ then by inspection of the algorithm $\mathbb{R} = \mathbb{R}''; \mathsf{CUnify}(\mathbb{R}''(\mathbb{C}_1'), \mathbb{R}'(\mathbb{C}_2'))$ where $\mathbb{R}'' = \mathsf{BUNIFY}(b, c)$. By completeness of BUNIFY $\mathbb{R}''$ is defined and most general (i.e. $\mathbb{R}'' \leq \mathbb{R}'$), and by the induction hypothesis $\mathsf{CUnify}(\mathbb{R}''(\mathbb{C}_1'), \mathbb{R}''(\mathbb{C}_2'))$ is also defined and produces a most general unifier. Then $\mathbb{R} = \mathsf{CUnify}(\mathbb{C}_1, \mathbb{C}_2)$ is defined, and $\mathbb{R} \leq \mathbb{R}'$ as required.

□

Theorem 5.2.18 (Termination of CUnify) *The algorithm* CUnify *terminates for all finite inputs.*

Proof. By induction on the length of the intersection of the domains of the inputs. There are two cases to consider; if the intersection is empty then the algorithm terminates. If there is at least one pair in the intersection the algorithm calls BUNIFY which terminates, then makes a recursive call on the remainder of the inputs. Since each step either terminates or reduces the size of the inputs, the algorithm eventually terminates.

For example, consider the case of $\mathsf{CUnify}(\{x : b\} \cup \mathbb{C}_1, \{x : c\} \cup \mathbb{C}_2)$: two steps are involved; $\mathbb{R} = \mathsf{BUNIFY}(b, c)$ terminates, and hence the recursive call on the remainder of the input $\mathsf{CUnify}(\mathbb{R}(\mathbb{C}_1), \mathbb{R}(\mathbb{C}_2))$ also terminates by the induction hypothesis. □

Theorem 5.2.19 (Soundness of $\mathbb{M}$)

$$\mathbb{R} = \mathbb{M}(b, \sigma^c) \Longrightarrow \exists d.\mathbb{R}(c) = \mathbb{R}(b \cdot d)$$

Proof. By definition of the algorithm and by cases on its inputs.

- $c = \mathrm{T}$: By definition $\mathbb{M}\left(b, \sigma^{\mathrm{T}}\right) = \mathsf{BUNIFY}(b, \mathrm{T})$ which by the statement is defined. Then by soundness of BUNIFY $\mathbb{R}(c) = \mathbb{R}(\mathrm{T}) = \mathrm{T} = \mathbb{R}(b) = \mathbb{R}(b \cdot d)$, where $d = \mathrm{T}$.

- $c = \mathrm{U}$: Then $\mathbb{R} = \mathbb{M}\left(b, \sigma^{\mathrm{U}}\right) = \mathsf{Id}$. Hence $\mathbb{R}(\mathrm{U}) = \mathrm{U} = \mathbb{R}(b \cdot d) = b \cdot d$, where $d = \mathrm{U}$.

- $c \neq \mathrm{T}$, $c \neq \mathrm{U}$: Then $\mathbb{R} = \mathbb{M}(b, \sigma^c) = \mathsf{BUNIFY}(c, b \cdot i)$ (where i is fresh) which from the statement is defined. Hence by soundness of BUNIFY $\mathbb{R}(c) = \mathbb{R}(b \cdot i) = \mathbb{R}(b \cdot d)$, where $d = i$.

- $\mathbb{M}\left(b, \sigma^c \overrightarrow{\sigma'^e}\right)$: By the algorithm definition $\mathbb{R} = \mathbb{R}'; \mathbb{M}\left(\mathbb{R}'(b), \mathbb{R}'(\overrightarrow{\sigma'^e})\right)$ where $\mathbb{R}' = \mathbb{M}(b, \sigma^c)$. By the induction hypothesis soundness holds for $\mathbb{R}'$ and for $\mathbb{M}\left(\mathbb{R}'(b), \mathbb{R}'(\overrightarrow{\sigma'^e})\right)$; hence it also holds for $\mathbb{R}'; \mathbb{M}\left(\mathbb{R}'(b), \mathbb{R}'(\overrightarrow{\sigma'^e})\right)$.

□

Theorem 5.2.20 (Completeness of $\mathbb{M}$) $\forall b, c, \sigma$. *if* $\exists d, \mathbb{R}'$ *such that* $\mathbb{R}'(c) = \mathbb{R}'(b \cdot d)$ *then* $\mathbb{R} = \mathbb{M}(b, \sigma^c)$ *is defined, and* $\mathbb{R} \leq \mathbb{R}'$.

Proof. By cases on the input and the definition of the algorithm.

- $c = \mathrm{T}$: By definition of the algorithm $\mathbb{M}\left(b, \sigma^{\mathrm{T}}\right) = \mathsf{BUNIFY}(b, \mathrm{T})$; then by completeness of BUNIFY $\mathbb{R}$ is defined and $\mathbb{R} \leq \mathbb{R}'$. Note that since $c = \mathrm{T}$, $\mathbb{R}'(c) = \mathrm{T} = \mathbb{R}'(b \cdot d)$ (where $d = \mathrm{T}$).

- $c = \mathrm{U}$: By definition of the algorithm $\mathbb{R} = \mathbb{M}\big(b, \sigma^{\mathrm{U}}\big) = \mathsf{Id}$ which is defined, and $\mathsf{Id} \leq \mathbb{R}'$ for all $\mathbb{R}'$.
- $c \neq \mathrm{T}$, $c \neq \mathrm{U}$: Then $\mathbb{R} = \mathbb{M}(b, \sigma^c) = \mathsf{BUNIFY}(c, b \cdot i)$ (where i is fresh). If this is defined then by soundness of BUNIFY $\mathbb{R}(c) = \mathbb{R}(b \cdot i)$; since $\mathbb{R}'(c) = \mathbb{R}'(b \cdot d)$ for some $\mathbb{R}'$ and d then $\mathbb{M}(b, \sigma^c) = \mathsf{BUNIFY}(c, b \cdot i)$ is defined by completeness of BUNIFY (noting that if necessary $\mathbb{R}(i) = d$). Also by completeness of BUNIFY, $\mathbb{R} \leq \mathbb{R}'$ as required.
- $\mathbb{M}\Big(b, \sigma^c \overrightarrow{\sigma'^e}\Big)$: By the algorithm definition, $\mathbb{R} = \mathbb{R}''; \mathbb{M}\Big(\mathbb{R}''(b), \mathbb{R}''(\overrightarrow{\sigma'^e})\Big)$ where $\mathbb{R}'' = \mathbb{M}(b, \sigma^c)$. By the induction hypothesis, the result holds for $\mathbb{R}''$ (that is, it is defined and $\mathbb{R}'' \leq \mathbb{R}'$) and also by the induction hypothesis completeness holds for $\mathbb{M}\Big(\mathbb{R}''(b), \mathbb{R}''(\overrightarrow{\sigma'^e})\Big)$; thus the result holds for $\mathbb{R} = \mathbb{R}''; \mathbb{M}\Big(\mathbb{R}''(b), \mathbb{R}''(\overrightarrow{\sigma'^e})\Big)$.

□

Theorem 5.2.21 (Termination of $\mathbb{M}$) *The algorithm $\mathbb{M}$ terminates for all finite inputs.*

Proof. By induction on the length of the inputs. Each step either terminates (explicitly, with value Id, or with a call to BUNIFY which terminates), or makes two recursive calls on subsets of the inputs. Since each step either terminates or reduces the size of the inputs, if the inputs are finite the algorithm must eventually terminate. □

5.3 Type Inference For System $\mathcal{T}_{\mathsf{FO}\pi}$

5.3.1 Implementation

Type inference for the plain first-order system is a relatively straight-forward matter of reading the rules of Figure 3.4 (page 57) in a bottom-up fashion (that is, recursively walking back up the deduction tree), using substitutions from the unification algorithms of Section 5.2 above to ensure that the results may be correctly combined.

The algorithm is shown in Figures 5.1. It takes as input a valid term in System $\mathcal{T}_{\mathsf{FO}\pi}$, and returns a pair of an environment assigning types to all free variables in the term such that the term is well-typed, and an accompanying substitution for all declared types in the term. Note that in the syntax, on Page 47, annotation variables are admitted but type variables are not, and the term is expected to conform to this.

The substitution is required for the cases where a declared type contains an annotation variable which is assigned a new value during unification. For example, consider the (useless) term

$$(\nu x : ((\,)^i)^{\mathrm{T}})\overline{x}[y].\mathbf{0}$$

Typing would proceed by assigning y the type α^j and x the type $(\alpha^j)^k$, for fresh α, j, k. Next, to form the restriction the declared type of x is unified with the deduced type; that is a substitution must be found to equate $(\alpha^j)^k$ with $((\,)^i)^{\mathrm{T}}$, which reduces to assigning T to k, $()$ to α, and equating i and j. Should the unification algorithm resolve that $i := j$, then the above term would no-longer be typed under the deduced environment of $y : ()^j$ unless i is replaced by j in the term. Hence a substitution to be applied to the term is also returned.[2]

In the algorithm, α is a fresh type variable, and i, j, k are fresh annotations.

5.3.2 Correctness of $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}$

Correctness is demonstrated via two separate results; a soundness result stating that every successful execution of the algorithm produces a valid type in system $\mathcal{T}_{\mathsf{FO}\pi}$, and a completeness result stating that if a valid typing exists for a term in system $\mathcal{T}_{\mathsf{FO}\pi}$ then the algorithm is defined, and furthermore produces a most-general typing (see Definition 5.2.3). Note that the combination, including the most-general result, implies the existence of principle types in the system.

Soundness

Theorem 5.3.1 (Soundness of $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}$)

$$\forall \mathrm{P}.\, \langle \Gamma, \mathbb{S} \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{P}) \Longrightarrow \Gamma \vdash_{\mathrm{FO}} \mathbb{S}(\mathrm{P}) : \mathsf{Proc}$$

Proof. By induction on the structure of P.

- $\mathrm{P} \equiv \mathbf{0}$: By inspection, $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathbf{0}) = \langle \emptyset, \mathsf{Id} \rangle$ and by the zero type rule $\emptyset \vdash_{(\mathrm{FO})} \mathbf{0} : \mathsf{Proc}$ as required.

[2] This is a similar procedure to that often seen in polymorphic type inference, although for slightly different reasons. An alternative approach would be to thread an environment through the algorithm, with the typed binders added in the recursive calls on the body. The difference in complexity of the algorithm and proofs is minimal and it seems a matter of taste, with the stand-alone feel of the chosen scheme preferred.

$$
\begin{array}{rcl}
\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathbf{0}) & = & \langle \emptyset, \mathsf{Id} \rangle \\
\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(!\mathrm{P}) & = & \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{P}) \\
\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}\big((\nu x : \sigma^b)\mathrm{P}\big) & = & \text{let } \langle \Gamma, \mathbb{S}_1 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{P}) \\
 & & \quad \mathbb{S}_2 = \mathsf{Unify}\big(\Gamma, \{x : \sigma^b\}\big) \\
 & & \text{in } \langle \mathbb{S}_2(\Gamma_x), \mathbb{S}_2 \rangle \\
\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{P}|\mathrm{Q}) & = & \text{let } \langle \Gamma, \mathbb{S}_1 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{P}) \\
 & & \quad \langle \Theta, \mathbb{S}_2 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathbb{S}_1(\mathrm{Q})) \\
 & & \quad \mathbb{S}_3 = \mathsf{Unify}(\mathbb{S}_2(\Gamma), \Theta) \\
 & & \text{in } \langle \mathbb{S}_3(\mathbb{S}_2(\Gamma), \Theta), \mathbb{S}_1; \mathbb{S}_2; \mathbb{S}_3 \rangle \\
\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{P} + \mathrm{Q}) & = & \text{let } \langle \Gamma, \mathbb{S}_1 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{P}) \\
 & & \quad \langle \Theta, \mathbb{S}_2 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathbb{S}_1(\mathrm{Q})) \\
 & & \quad \mathbb{S}_3 = \mathsf{Unify}(\mathbb{S}_2(\Gamma), \Theta) \\
 & & \text{in } \langle \mathbb{S}_3(\mathbb{S}_2(\Gamma), \Theta), \mathbb{S}_1; \mathbb{S}_2; \mathbb{S}_3 \rangle \\
\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\overline{x}[\overrightarrow{y}].\mathrm{P}) & = & \text{let } \langle \Gamma, \mathbb{S}_1 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{P}) \\
 & & \quad \mathbb{S}_2 = \mathsf{Unify}\Big(\Gamma, \{x : (\overrightarrow{\alpha^i})^j, \overrightarrow{y} : \overrightarrow{\alpha^i}\}\Big) \\
 & & \quad \mathbb{S}_3 = \mathbb{S}_2; \mathbb{M}\Big(\mathbb{S}_2(j), \mathbb{S}_2(\overrightarrow{\alpha^i})\Big) \\
 & & \text{in } \Big\langle \mathbb{S}_3\left(\Gamma, x : (\overrightarrow{\alpha^i})^j, \overrightarrow{y} : \overrightarrow{\alpha^i}\right), \mathbb{S}_3 \Big\rangle \\
\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}\Big(x(\overrightarrow{y} : \overrightarrow{\sigma^b}).\mathrm{P}\Big) & = & \text{let } \langle \Gamma, \mathbb{S}_1 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{P}) \\
 & & \quad \mathbb{S}_2 = \mathsf{Unify}\Big(\Gamma, \{x : (\overrightarrow{\sigma^b})^j, \overrightarrow{y} : \overrightarrow{\sigma^b}\}\Big) \\
 & & \quad \mathbb{S}_3 = \mathbb{S}_2; \mathbb{M}\Big(\mathbb{S}_2(j), \mathbb{S}_2(\overrightarrow{\sigma^i})\Big) \\
 & & \text{in } \mathbb{S}_3 \Big\langle \left(\Gamma_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^b})^j\right), \mathbb{S}_3 \Big\rangle \\
\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}\big(x(y : \sigma^b)_{\mathrm{certify}}\mathrm{P} \oplus \mathrm{Q}\big) & = & \text{let } \langle \Gamma, \mathbb{S}_1 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{P}) \\
 & & \quad \langle \Theta, \mathbb{S}_2 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathbb{S}_1(\mathrm{Q})) \\
 & & \quad \mathbb{S}_3 = \mathbb{S}_1; \mathbb{S}_2; \mathsf{Unify}\big(\mathbb{S}_2(\Gamma), \\
 & & \qquad\qquad \mathbb{S}_1; \mathbb{S}_2(\{x : (\sigma^b)^j, y : \sigma^{\mathrm{T}}\})\big) \\
 & & \quad \mathbb{S}_4 = \mathbb{S}_3; \mathsf{Unify}\big(\mathbb{S}_3(\Theta), \mathbb{S}_3(\{x : (\sigma^b)^j, y : \sigma^{\mathrm{U}}\})\big) \\
 & & \quad \mathbb{S}_5 = \mathbb{S}_4; \mathsf{Unify}(\mathbb{S}_4(\Gamma_y), \mathbb{S}_4(\Theta_y)) \\
 & & \quad \mathbb{S}_6 = \mathbb{S}_5; \mathbb{M}\big(\mathbb{S}_5(j), \mathbb{S}_5(\sigma^b)\big) \\
 & & \text{in } \big\langle \mathbb{S}_6\left(\Gamma_y, \Theta_y, x : (\sigma^b)^j\right), \mathbb{S}_6 \big\rangle
\end{array}
$$

Figure 5.1: Type Inference for System $\mathcal{T}_{\mathsf{FO}\pi}$

- P $\equiv$!Q: By inspection, $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(!\mathrm{Q}) = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{Q})$. By the induction hypothesis, if $\langle \Gamma, \mathbb{S} \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{Q})$ then $\Gamma \vdash_{\mathrm{FO}} \mathbb{S}(\mathrm{Q}) : \mathsf{Proc}$; hence by the replication type rule $\langle \Gamma, \mathbb{S} \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(!\mathrm{Q}) \Longrightarrow \Gamma \vdash_{\mathrm{FO}} \mathbb{S}(!\mathrm{Q}) : \mathsf{Proc}$ as required.

- P $\equiv (\nu x : \sigma^b)$Q: By the induction hypothesis soundness holds for Q; that is if $\langle \Gamma, \mathbb{S}_1 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{Q})$ then $\Gamma \vdash_{\mathrm{FO}} \mathbb{S}_1(\mathrm{Q}) : \mathsf{Proc}$. By the soundness of Unify, $\mathbb{S}_2 = \mathsf{Unify}\big(\Gamma, \{x : \sigma^b\}\big)$ implies $\mathbb{S}_2(\Gamma(x)) = \mathbb{S}_2(\sigma^b)$ (if $x \in \mathrm{dom}\Gamma$). Hence by the abstraction type rule and inspection of the algorithm, $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}\big((\nu x : \sigma^b)\mathrm{Q}\big) = \langle \mathbb{S}_2(\Gamma_x), \mathbb{S}_2 \rangle$ and $\mathbb{S}_2(\Gamma_x) \vdash_{\mathrm{FO}} \mathbb{S}_2((\nu x : \sigma_b)\mathrm{Q}) : \mathsf{Proc}$ as required.

- P $\equiv$ Q|R: By the induction hypothesis soundness holds for Q and R individually; that is $\langle \Gamma, \mathbb{S}_1 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{Q}) \Longrightarrow \Gamma \vdash_{\mathrm{FO}} \mathbb{S}_1(\mathrm{Q}) : \mathsf{Proc}$ and $\langle \Theta, \mathbb{S}_2 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathbb{S}_1(\mathrm{R})) \Longrightarrow \Theta \vdash_{\mathrm{FO}} \mathbb{S}_1; \mathbb{S}_2(\mathrm{R}) : \mathsf{Proc}$. Then by soundness of Unify if $\mathbb{S}_3 = \mathsf{Unify}(\mathbb{S}_2(\Gamma), \Theta)$ then $\mathbb{S}_2; \mathbb{S}_3(\Gamma) \asymp \mathbb{S}_3(\Theta)$ and hence $\mathbb{S}_3(\mathbb{S}_2(\Gamma), \Theta) \vdash_{\mathrm{FO}} \mathbb{S}1; \mathbb{S}_2; \mathbb{S}_3(\mathrm{P}|\mathrm{Q}) : \mathsf{Proc}$ as required.

- P $\equiv$ Q + R: Similarly.

- P $\equiv \overline{x}[\overrightarrow{y}]$.Q: By the induction hypothesis soundness holds for Q individually; that is $\langle \Gamma, \mathbb{S}_1 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{Q}) \Longrightarrow \Gamma \vdash_{\mathrm{FO}} \mathbb{S}_1(\mathrm{Q}) : \mathsf{Proc}$. By the soundness of Unify the following holds:

$$\mathbb{S}_2 = \mathsf{Unify}\Big(\Gamma, \{x : (\overrightarrow{\alpha^i})^j, \overrightarrow{y} : \overrightarrow{\alpha^i}\}\Big) \Longrightarrow$$
$$\forall y : \alpha^i \in \mathrm{dom}\Gamma \cap \{x : (\overrightarrow{\alpha^i})^j, \overrightarrow{y} : \overrightarrow{\alpha^i}\}.\mathbb{S}_2(\Gamma(y)) = \mathbb{S}_2(\alpha^i)$$

  (where $\overrightarrow{\alpha^i}$ and j are all fresh variables). Note that this also unifies the object of the type of x with the types of $\overrightarrow{y}$. Lastly, it must be ensured that $\overrightarrow{\alpha^i}$ has the form $c \cdot \overrightarrow{\sigma^b}$ as required by the well-formed type rule; by the soundness of $\mathbb{M}$: $\mathbb{S}_3 = \mathbb{S}_2; \mathbb{M}\Big(\mathbb{S}_2(j), \mathbb{S}_2(\overrightarrow{\alpha^i})\Big)$ implies that for all α^i in $\overrightarrow{\alpha^i}$ then $\mathbb{S}_3(i) = \mathbb{S}_3(j \cdot k)$ for some k as required. Hence $\mathbb{S}_3(\Gamma, x : (\alpha^i)^j, \overrightarrow{y} : \overrightarrow{\alpha^i} \vdash_{(\mathrm{FO})} \overline{x}[\overrightarrow{y}].\mathrm{Q} : \mathsf{Proc})$ as required.

- P $\equiv x(\overrightarrow{y} : \overrightarrow{\sigma^b})$.Q: The case for input proceeds similarly to the case for output, with the exception that the types of the argument names are specified in the term. By the induction hypothesis soundness holds for Q individually; that is $\langle \Gamma, \mathbb{S}_1 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{Q}) \Longrightarrow \Gamma \vdash_{\mathrm{FO}} \mathbb{S}_1(\mathrm{Q}) : \mathsf{Proc}$.

By the soundness of Unify the following holds:

$$\begin{aligned}\mathbb{S}_2 = \mathsf{Unify}\Big(\Gamma, \{x : (\overrightarrow{\sigma^b})^j, \overrightarrow{y} : \overrightarrow{\sigma^b}\}\Big) \implies \\ \mathbb{S}_2(\Gamma(x)) = \mathbb{S}_2(x : (\overrightarrow{\sigma^b})^j) \quad \text{and} \\ \forall y_i : \sigma_i^{b_i} \in \overrightarrow{y} : \overrightarrow{\sigma^b}.\mathbb{S}_2(\Gamma(y_i)) = \mathbb{S}_2(\sigma_i^{b_i})\end{aligned}$$

(where j is a fresh variable, and assuming in each case that the name is in the domain of Γ — the statement is obviously redundant otherwise). Lastly, to ensure the well-formed type rule holds: by the soundness of $\mathbb{M}$, $\mathbb{S}_3 = \mathbb{S}_2; \mathbb{M}\Big(\mathbb{S}_2(j), \mathbb{S}_2(\overrightarrow{\sigma^i})\Big)$ implies that for all σ^b in $\overrightarrow{\sigma^b}$ then $\mathbb{S}_3(b) = \mathbb{S}_3(j \cdot c)$ for some c as required. Hence

$$\mathbb{S}_3(\Gamma_{\overrightarrow{y}}, x : (\sigma^b)^j \vdash_{(\mathrm{FO})} x(\overrightarrow{y} : \sigma^b).\mathrm{Q} : \mathsf{Proc})$$

as required.

- $\mathrm{P} \equiv x(y : \sigma^i)_{\underset{\text{certify}}{?}}\mathrm{Q} \oplus \mathrm{R}$: By the induction hypothesis soundness holds for Q and R individually; that is

$$\begin{aligned}\langle \Gamma, \mathbb{S}_1 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{Q}) \implies \Gamma \vdash_{\mathrm{FO}} \mathbb{S}_1(\mathrm{Q}) : \mathsf{Proc} \\ \text{and} \quad \langle \Theta, \mathbb{S}_2 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathbb{S}_1(\mathrm{R})) \implies \Theta \vdash_{\mathrm{FO}} \mathbb{S}_1; \mathbb{S}_2(\mathrm{R}) : \mathsf{Proc}\end{aligned}$$

Then by soundness of Unify,

$$\begin{aligned}\mathbb{S}_3 = \mathbb{S}_1; \mathbb{S}_2; \mathsf{Unify}\big(\mathbb{S}_2(\Gamma), \mathbb{S}_1; \mathbb{S}_2(\{x : (\sigma^b)^j, y : \sigma^{\mathrm{T}}\})\big) \implies \\ \mathbb{S}_3(\Gamma) \asymp \mathbb{S}_3(\{x : (\sigma^b)^j, y : \sigma^{\mathrm{T}}\})\end{aligned}$$

and

$$\begin{aligned}\mathbb{S}_4 = \mathbb{S}_3; \mathsf{Unify}\big(\mathbb{S}_3(\Theta), \mathbb{S}_3(\{x : (\sigma^b)^j, y : \sigma^{\mathrm{U}}\})\big) \implies \\ \mathbb{S}_4(\Theta) \asymp \mathbb{S}_4(\{x : (\sigma^b)^j, y : \sigma^{\mathrm{U}}\})\end{aligned}$$

Similarly, $\mathbb{S}_5 = \mathbb{S}_4; \mathsf{Unify}(\mathbb{S}_4(\Gamma_y), \mathbb{S}_4(\Theta_y))$ implies $\mathbb{S}_5(\Gamma_y) \asymp \mathbb{S}_5(\Theta_y)$ (note that this unification must be performed in the absence of y, owing to the conflicting types in each environment). Note that by the well-formed type rule $b = i \cdot j$ for some i and j. This is ensured by the soundness of $\mathbb{M}$; where $\mathbb{S}_6 = \mathbb{S}_5; \mathbb{M}\big(\mathbb{S}_5(j), \mathbb{S}_5(\sigma^b)\big)$ (for some fresh variable j) implies $\mathbb{S}_6(b) = \mathbb{S}_6(j \cdot k)$ for some k. Hence by the certify rule

$$\mathbb{S}_6(\Gamma_y, \Theta_y, x : (\sigma^b)^j \vdash_{(\mathrm{FO})} x(y : \sigma^b)_{\underset{\text{certify}}{?}}\mathrm{Q} \oplus \mathrm{R} : \mathsf{Proc})$$

as required.

□

Completeness

The reader is referred back to page 128 for the definition of $\Gamma \leq \Gamma'$ if necessary.

Theorem 5.3.2 (Completeness of $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}$) $\forall \Gamma'$, P. *If there is a valid deduction in System $\mathcal{T}_{\mathsf{FO}\pi}$ for $\Gamma' \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}$ then $\langle \Gamma, \mathbb{S} \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{P})$ is defined, and $\Gamma \leq \Gamma'$*

Proof. By cases on the structure of P, then by inference on the derivation of $\Gamma' \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}$ and by inspection of the algorithm.

- $\Gamma' \vdash_{\mathrm{FO}} \mathbf{0} : \mathsf{Proc}$: By the zero type rule (and as many applications of the weaken rule as necessary) this is valid for any Γ'. By inspection, $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathbf{0})$ is defined and $\langle \emptyset, \mathsf{Id} \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathbf{0})$. Then clearly $\emptyset \leq \Gamma$ as $\emptyset \subseteq \Gamma'$ for all Γ', as required.

- $\Gamma' \vdash_{\mathrm{FO}} !\mathrm{P} : \mathsf{Proc}$: By inspection, $\langle \Gamma, \mathbb{S} \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(!\mathrm{P}) = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{P})$ and by the induction hypothesis completeness holds for P: that is, given a valid deduction $\Gamma' \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}$ then $\langle \Gamma, \mathbb{S} \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{P})$ is defined and $\Gamma \leq \Gamma'$. Hence by the replication rule completeness holds for !P.

- $\Gamma', \Theta', \Delta \vdash_{\mathrm{FO}} \mathrm{P}|\mathrm{Q} : \mathsf{Proc}$: This deduction must have ended in zero or more uses of the weaken rule, preceded by an instance of the composition rule with antecedents $\Gamma' \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}$ and $\Theta' \vdash_{\mathrm{FO}} \mathrm{Q} : \mathsf{Proc}$ (where the domain of Δ contains all names introduced through the final uses of the weaken rule). By the induction hypothesis completeness holds for P and Q individually; that is $\langle \Gamma, \mathbb{S}_1 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{P})$ and $\langle \Theta, \mathbb{S}_1 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathbb{S}_1(\mathrm{Q}))$ are both defined, with $\Gamma \leq \Gamma'$ and $\Theta \leq \Theta'$. Then by completeness of Unify, if $\mathbb{S}_3 = \mathsf{Unify}(\mathbb{S}_2(\Gamma), \Theta)$ then $\mathbb{S}_3(\mathbb{S}_2(\Gamma), \Theta) \leq \Gamma', \Theta'$ as required.

- $\Gamma' \vdash_{(\mathrm{FO})} \mathrm{P} + \mathrm{Q} : \mathsf{Proc}$: Similarly.

- $\Gamma' \vdash_{(\mathrm{FO})} x(\overrightarrow{y} : \overrightarrow{\sigma^b}).\mathrm{P} : \mathsf{Proc}$: The deduction must have ended in zero or more uses of the weaken rule, preceded by an instance of the input rule with antecedent

$$\Gamma'', \overrightarrow{y} : \overrightarrow{\sigma^b}, x : (\overrightarrow{\sigma^b})^c \vdash_{\mathrm{FO}} \mathrm{P} : \mathsf{Proc}$$

  (where $\Gamma'', x : (\overrightarrow{\sigma^b})^c \subseteq \Gamma'$; assuming for notational convenience that $\overrightarrow{y} \notin \mathrm{dom}\Gamma'$). By the induction hypothesis completeness holds for P

individually: $\langle \Gamma, \mathbb{S}_1 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{Q})$ is defined, and $\Gamma \leq \Gamma', x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma^b}$. Then by Theorem 5.2.14 $\mathbb{S}_2 = \mathsf{Unify}\Big(\Gamma, \{x : (\overrightarrow{\sigma^b})^j, \overrightarrow{y} : \overrightarrow{\sigma^b}\}\Big)$ is complete (implying that $\mathbb{S}_2(\Gamma, x : (\overrightarrow{\sigma^b})^j, \overrightarrow{y} : \overrightarrow{\sigma^b}) \leq \Gamma'', x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma^b}$). Likewise

$$\mathbb{S}_3 = \mathbb{S}_2; \mathbb{M}\Big(\mathbb{S}_2(j), \mathbb{S}_2(\overrightarrow{\sigma^i})\Big)$$

is also complete; implying $\mathbb{S}_3(b) = j \cdot k$ for some k and each $\sigma^b \in \overrightarrow{\sigma^b}$. Hence (noting that the weaken rule increases the domain of Γ', so the requirement that $\mathrm{dom}\Gamma \subseteq \Gamma'$ is not affected):

$$\mathbb{S}_3(\Gamma_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^b})^j) \leq \Gamma'$$

as required.

- $\Gamma' \vdash_{(\mathrm{FO})} \overline{x}[\overrightarrow{y}].\mathrm{P} : \mathsf{Proc}$: Similarly; without the need to remove $\overrightarrow{y}$ from the environment of the consequent, and using fresh variables for the types of $\overrightarrow{y}$.

- $\Gamma', \Theta', x : (\sigma^i)^c, \Delta \vdash_{(\mathrm{FO})} x(y : \sigma^i)_{\mathsf{certify}}\mathrm{P} \oplus \mathrm{Q} : \mathsf{Proc}$: This deduction must have ended in zero or more uses of the weaken rule, preceded by an instance of the certify rule with antecedents

$$\begin{aligned} \Gamma', x : (\sigma^i)^c, y : \sigma^{\mathrm{T}} \quad &\vdash_{\mathrm{FO}} \quad \mathrm{P} : \mathsf{Proc} \\ \text{and} \quad \Theta', x : (\sigma^i)^c, y : \sigma^{\mathrm{U}} \quad &\vdash_{\mathrm{FO}} \quad \mathrm{Q} : \mathsf{Proc} \end{aligned}$$

where the domain of Δ is all the names introduced through the weaken rule, and assuming for notational convenience that $y \notin \mathrm{dom}\Gamma' \cap \Theta'$. By the induction hypothesis, completeness holds for P and Q individually; that is $\langle \Gamma, \mathbb{S}_1 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{P})$ and $\langle \Theta, \mathbb{S}_2 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathbb{S}_1(\mathrm{Q}))$ are both defined, with $\Gamma \leq \Gamma', x : (\sigma^{bc})^c, y : \sigma^{\mathrm{T}}$ and $\Theta \leq \Theta', x : (\sigma^{bc})^c, y : \sigma^{\mathrm{U}}$. By completeness of Unify, $\mathbb{S}_3 = \mathbb{S}_1; \mathbb{S}_2; \mathsf{Unify}\big(\mathbb{S}_2(\Gamma), \mathbb{S}_1; \mathbb{S}_2(\{x : (\sigma^b)^j, y : \sigma^{\mathrm{T}}\})\big)$ is complete, and similarly so is $\mathbb{S}_4 = \mathbb{S}_3; \mathsf{Unify}\big(\mathbb{S}_3(\Theta), \mathbb{S}_3(\{x : (\sigma^b)^j, y : \sigma^{\mathrm{U}}\})\big)$. Again by completeness of Unify, $\mathbb{S}_5 = \mathbb{S}_4; \mathsf{Unify}\big(\mathbb{S}_4(\Gamma_y), \mathbb{S}_4(\Theta_y)\big)$ is complete (note that the unification must be performed with y removed from each domain, due to the conflicting types in each branch). By completeness of $\mathbb{M}$, $\mathbb{S}_6 = \mathbb{S}_5; \mathbb{M}\big(\mathbb{S}_5(j), \mathbb{S}_5(\sigma^b)\big)$ is defined and most general. Hence

$$\mathbb{S}_7\left(\Gamma_y, \Theta_y, x : (\sigma^b)^j\right) \leq \Gamma'_y, \Theta'_y, x : (\sigma^{cb})^c$$

as required.

□

5.3.3 Termination of $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}$

As an additional stage in analysing the algorithm, it is desirable to show that it always terminates (lack of termination in all cases does not imply lack of correctness, none the less it is a useful result to establish).

Theorem 5.3.3 (Termination of $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}$) *The algorithm $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}$ terminates for all valid finite inputs.*

Proof. By induction on the structure of the input. Each step either: terminates immediately (returning a result); calls one of the algorithms Unify or $\mathbb{M}$ which terminates by Theorems 5.2.15 and 5.2.21; or makes a recursive call on a subset of the input. Since each stage either terminates explicitly or recurses on a subset of the input, if the input is finite then the algorithm must eventually terminate.

As an example, consider the case of $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}\Big(x(\overrightarrow{y} : \overrightarrow{\sigma^b}).\mathrm{P}\Big)$: there are three stages involved, all of which can be shown to terminate. Firstly, $x(\overrightarrow{y} : \overrightarrow{\sigma^b}).\mathrm{P}$ is finite from the statement, so the recursive call $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}(\mathrm{P})$ must also have finite input, and thus terminates by the induction hypothesis. Likewise, the sequence $\overrightarrow{y}$ is also finite according to the statement, so the call $\mathbb{S}_2 = \mathsf{Unify}\Big(\Gamma, \{x : (\overrightarrow{\sigma^b})^j, \overrightarrow{y} : \overrightarrow{\sigma^b}\}\Big)$ must terminate by Theorem 5.2.15. Finally, by Theorem 5.2.21 the call $\mathbb{S}_3 = \mathbb{S}_2; \mathbb{M}\Big(\mathbb{S}_2(j), \mathbb{S}_2(\overrightarrow{\sigma^i})\Big)$ also terminates, and as each individual stage terminates the algorithm for $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}\Big(x(\overrightarrow{y} : \overrightarrow{\sigma^b}).\mathrm{P}\Big)$ also terminates. □

5.4 Type Inference For System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$

Analogously to Igarashi and Kobayashi (2000) the rules of Figure 3.4 (page 57) alone are insufficient for type inference as it is also necessary to encode information about the relationships between annotation expressions, in the form of constraints (Definition 5.4.1). An inferred type can then take the form of an environment and substitution as before, and a set of constraints on the annotations in that environment.

Definition 5.4.1 (Annotation Constraints) *(This material largely follows Igarashi and Kobayashi (2000)).*

1. *Constraints are represented by $\mathcal{C}$, and are a set of expressions of the form $b \leq_{\mathsf{B}} c$.*

2. *A substitution* $\mathbb{S}$ *(or* $\mathbb{R}$*, since constraints are composed solely from annotations) is said to* solve $\mathcal{C}$ *if and only if* $\forall(b \leq_{\mathsf{B}} c) \in \mathcal{C}.\mathbb{S}(b) \leq_{\mathsf{B}} \mathbb{S}(c)$*, and* $\mathbb{S}(b)$ *is a constant expression and/or* $\mathbb{S}(c)$ *is a constant expression. For example, if* $\mathcal{C} = \{i \leq_{\mathsf{B}} j\}$ *then* $\mathbb{S}_1 = \mathsf{Id}[i := \mathrm{U}][j := \mathrm{T}]$*,* $\mathbb{S}_2 = \mathsf{Id}[j := \mathrm{T}]$*, and* $\mathbb{S}_3 = \mathsf{Id}[i := \mathrm{U}][j := k \cdot \mathrm{T} + \overline{k} \cdot \mathrm{U}]$ *are all solutions of* $\mathcal{C}$*; while* $\mathbb{S}_4 = \mathsf{Id}[j := k]$ *and* $\mathbb{S}_5 = \mathsf{Id}[i := \mathrm{T}][j := \mathrm{U}]$ *are not.*

3. *The operator ',' is defined as shorthand for the union operator as usual.*

4. *Constraint sets are only distinguished up to non-trivial constraints; for example* $\mathcal{C}_1 = \{i \leq_{\mathsf{B}} j\}$ *is not distinguished from* $\mathcal{C}_2 = \{i \leq_{\mathsf{B}} j, \mathrm{U} \leq_{\mathsf{B}} \mathrm{T}, j \leq_{\mathsf{B}} \mathrm{T}\}$.

5. *Write* $\mathcal{C} \leq \mathcal{C}'$ *if and only if every solution of* $\mathcal{C}$ *is also a solution of* $\mathcal{C}'$*. For example,* $\{i \leq_{\mathsf{B}} j, j \leq_{\mathsf{B}} k\} \leq \{i \leq_{\mathsf{B}} k\}$ *(for instance, the substitution* $\mathsf{Id}[i := \mathrm{T}][j := \mathrm{T}]$ *solves both, but* $\mathsf{Id}[k := \mathrm{T}]$ *only solves the second).*

5.4.1 Implementation

The algorithm itself is shown in Figures 5.2 and 5.3. It takes as input a valid term in System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$ and returns a four-tuple consisting of an environment, a security level (annotation), a set of constraints on annotations, and a substitution to be applied to the term.

This particular form of algorithm is based on the work of Igarashi and Kobayashi (2000), who first described type reconstruction algorithms for linear type systems of the π-calculus (with input/output sub-typing).

5.4.2 Correctness of $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}$

As with $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}$, correctness of the algorithm $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}$ is demonstrated over two stages: a soundness result stating that all results returned by the algorithm produce a valid typing in the inference rules for system $\mathcal{T}_{\mathsf{FO}'\pi\leq}$, and a completeness result stating that if a valid deduction exists in System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$ for a given term then $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}$ is defined and the result returned is most general. Additionally, it must be shown that all constraints generated are consistent with the deduction; this is expressed in the second consequent of Theorem 5.4.2. To verify the validity of the constraints returned the function $\mathsf{Constraints}$ is defined over a deduction *tree* (the argument is written as a judgement rather than, say, a pair of an environment and a term, to emphasise the fact that it refers to a single node in a tree, with all the

$$
\begin{array}{rcl}
\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathbf{0}) & = & \langle \emptyset, i, \emptyset, \mathsf{Id} \rangle \\
\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}\big((\nu x : \sigma^b)\mathrm{P}\big) & = & \text{let } \langle \Gamma, c, \mathcal{C}, \mathbb{S}_1 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathrm{P}) \text{ in} \\
 & & \quad \mathbb{S}_2 = \mathbb{S}_1; \mathsf{Unify}\big(\Gamma, \mathbb{S}_1(\{x : \sigma^b\})\big) \\
 & & \text{in } \langle \mathbb{S}_2(\Gamma_x), \mathbb{S}_2(c), \mathbb{S}_2(\mathcal{C}), \mathbb{S}_2 \rangle \\
\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(!\mathrm{P}) & = & \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathrm{P}) \\
\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathrm{P} + \mathrm{Q}) & = & \text{let } \langle \Gamma, b, \mathcal{C}_1, \mathbb{S}_1 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathrm{P}) \\
 & & \quad \langle \Theta, c, \mathcal{C}_2, \mathbb{S}_2 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathbb{S}_1(\mathrm{Q})) \\
 & & \quad \mathbb{S}_3 = \mathbb{S}_2; \mathsf{Unify}(\mathbb{S}_2(\Gamma), \Theta) \\
 & & \text{in } \langle \mathbb{S}_3(\Gamma, \Theta), \mathbb{S}_3(b \cdot c), \mathbb{S}(\mathcal{C}_1, \mathcal{C}_2), \mathbb{S}_3 \rangle \\
\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathrm{P}|\mathrm{Q}) & = & \text{let } \langle \Gamma, b, \mathcal{C}_1, \mathbb{S}_1 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathrm{P}) \\
 & & \quad \langle \Theta, c, \mathcal{C}_2, \mathbb{S}_2 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathbb{S}_1(\mathrm{Q})) \\
 & & \quad \mathbb{S}_3 = \mathbb{S}_2; \mathsf{Unify}(\mathbb{S}_2(\Gamma), \Theta) \\
 & & \text{in } \langle \mathbb{S}_3(\Gamma, \Theta), \mathbb{S}_3(b \cdot c), \mathbb{S}(\mathcal{C}_1, \mathcal{C}_2), \mathbb{S}_3 \rangle
\end{array}
$$

Figure 5.2: Type Inference Algorithm for System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$

$$
\begin{aligned}
&\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}\Big(x(\overrightarrow{y} : \overrightarrow{\sigma^b}).\mathrm{P}\Big) = \\
&\quad \text{let } \langle \Gamma, c, \mathcal{C}, \mathbb{S}_1 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathrm{P}) \\
&\qquad \mathbb{S}_2 = \mathbb{S}_1; \mathsf{Unify}\Big(\Gamma, \mathbb{S}_1(\{x : (\overrightarrow{\sigma^j})^i, \overrightarrow{y} : \overrightarrow{\sigma^b}\})\Big) \\
&\qquad \mathbb{S}_3 = \mathbb{S}_2; \mathbb{M}\Big(\mathbb{S}_2(i), \mathbb{S}_2(\overrightarrow{\sigma^j})\Big) \\
&\quad \text{in } \Big\langle \mathbb{S}_3(\Gamma_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^j})^i), k, \\
&\qquad\quad \mathbb{S}_3(\mathcal{C} \cup \{\overrightarrow{b} \leq_{\mathsf{B}} \overrightarrow{j}, k \leq_{\mathsf{B}} c, k \leq_{\mathsf{B}} i\}), \mathbb{S}_3 \Big\rangle \\
&\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\overline{x}[\overrightarrow{y}].\mathrm{P}) = \\
&\quad \text{let } \langle \Gamma, c, \mathcal{C}, \mathbb{S}_1 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathrm{P}) \\
&\qquad \mathbb{S}_2 = \mathbb{S}_1; \mathsf{Unify}\Big(\Gamma, \{x : (\overrightarrow{\alpha^j})^i, \overrightarrow{y} : \overrightarrow{\alpha^k}\}\Big) \\
&\qquad \mathbb{S}_3 = \mathbb{S}_2; \mathbb{M}\Big(\mathbb{S}_2(i), \mathbb{S}_2(\overrightarrow{\alpha^j})\Big) \\
&\quad \text{in } \Big\langle \mathbb{S}_3(\Gamma \cup \{x : (\overrightarrow{\alpha^j})^i, \overrightarrow{y} : \overrightarrow{\alpha^k}\}), \mathbb{S}_3(l), \\
&\qquad\quad \mathbb{S}_3(\mathcal{C} \cup \{\overrightarrow{j} \leq_{\mathsf{B}} \overrightarrow{k}, l \leq_{\mathsf{B}} c, l \leq_{\mathsf{B}} i\}), \mathbb{S}_3 \Big\rangle \\
&\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}\big(x(y : \sigma^b)_{\text{certify}}\mathrm{P} \oplus \mathrm{Q}\big) = \\
&\quad \text{let } \langle \Gamma, c, \mathcal{C}_1, \mathbb{S}_1 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathrm{P}) \\
&\qquad \langle \Theta, d, \mathcal{C}_2, \mathbb{S}_2 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathbb{S}_1(\mathrm{Q})) \\
&\qquad \mathbb{S}_3 = \mathbb{S}_1; \mathbb{S}_2; \mathsf{Unify}\big(\mathbb{S}_2(\Gamma), \mathbb{S}_1; \mathbb{S}_2(\{x : (\sigma^b)^i, y : \sigma^{\mathrm{T}}\})\big) \\
&\qquad \mathbb{S}_4 = \mathbb{S}_3; \mathsf{Unify}\big(\mathbb{S}_3(\Theta), \mathbb{S}_3(\{x : (\sigma^b)^i, y : \sigma^{\mathrm{U}}\})\big) \\
&\qquad \mathbb{S}_5 = \mathbb{S}_4; \mathsf{Unify}\big(\mathbb{S}_4(\Gamma_y), \mathbb{S}_4(\Theta_y)\big) \\
&\qquad \mathbb{S}_6 = \mathbb{S}_5; \mathbb{M}\big(\mathbb{S}_5(i), \mathbb{S}_5(\sigma^b)\big) \\
&\quad \text{in } \big\langle \mathbb{S}_6(\Gamma_y, \Theta_y, x : (\sigma^b)^i), bj + \bar{b}k, \\
&\qquad\quad \mathbb{S}_6(\mathcal{C}_1, \mathcal{C}_2, j \leq_{\mathsf{B}} c, j \leq_{\mathsf{B}} i, k \leq_{\mathsf{B}} d, k \leq_{\mathsf{B}} i), \mathbb{S}_6 \big\rangle
\end{aligned}
$$

Figure 5.3: Type Inference Algorithm for System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$(continued)

tree known); it simply recursively walks the tree, collecting all constraints together. The definition is in Figure 5.4.

Soundness

Theorem 5.4.2 (Soundness of $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}$) *For any valid term* P *in System* $\mathcal{T}_{\mathsf{FO}'\pi\leq}$*,*

$$\langle \Gamma, b, \mathcal{C}, \mathbb{S} \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathrm{P})$$

implies that there is a derivation for

$$\Gamma \vdash^{b}_{\mathrm{FO}'\leq} \mathbb{S}(\mathrm{P}) : \mathsf{Proc}$$

and that

$$\mathsf{Constraints}\big(\Gamma \vdash^{b}_{\mathrm{FO}'\leq} \mathbb{S}(\mathrm{P}) : \mathsf{Proc}\big) = \mathcal{C}$$

Proof. By deduction on the structure of P and examination of the algorithm.

- $\mathrm{P} \equiv \mathbf{0}$: From the algorithm, $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathbf{0}) = \langle \emptyset, i, \emptyset, \mathsf{Id} \rangle$, and by the zero type rule $\emptyset \vdash^{i}_{(\mathrm{FO}'\leq)} \mathbf{0} : \mathsf{Proc}$ for all i, and $\mathsf{Constraints}\Big(\emptyset \vdash^{i}_{(\mathrm{FO}'\leq)} \mathbf{0} : \mathsf{Proc}\Big) = \emptyset$ as required.

- $\mathrm{P} \equiv !\mathrm{Q}$: From the algorithm, $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(!\mathrm{Q}) = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathrm{Q})$, and by the induction hypothesis soundness holds for Q and hence !Q by the replication type rule as required.

- $\mathrm{P} \equiv (\nu x)\mathrm{Q}$: By the induction hypothesis, $\langle \Gamma, c, \mathcal{C}, \mathbb{S}_1 \rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathrm{Q})$ implies $\Gamma \vdash^{c}_{\mathrm{FO}'\leq} \mathbb{S}_1(\mathrm{Q}) : \mathsf{Proc}$ and $\mathsf{Constraints}\Big(\Gamma \vdash^{c}_{\mathrm{FO}'\leq} \mathrm{Q} : \mathsf{Proc}\Big) = \mathcal{C}$. By the soundness of Unify, $\mathbb{S}_2 = \mathbb{S}_1; \mathsf{Unify}\big(\Gamma, \mathbb{S}_1(\{x : \sigma^b\})\big)$ implies $\mathbb{S}_2(\Gamma) \asymp \mathbb{S}_2(\{x : \sigma^b\})$ so by the restriction rule

  $$\Gamma_x \vdash^{b}_{\mathrm{FO}'\leq} \mathbb{S}_2((\nu x : \sigma^b)\mathrm{Q}) : \mathsf{Proc}$$

  and by definition

  $$\mathsf{Constraints}\big(\Gamma_x \vdash^{c}_{\mathrm{FO}'\leq} (\nu x : \sigma^b)\mathrm{Q} : \mathsf{Proc}\big) = \mathcal{C}$$

  as required.

$$\mathsf{Constraints}\Big(\Gamma \vdash^{b}_{\mathrm{FO}'\leq} \mathbf{0} : \mathsf{Proc}\Big) = \emptyset$$

$$\mathsf{Constraints}\Big(\Gamma \vdash^{b}_{\mathrm{FO}'\leq} !\mathrm{P} : \mathsf{Proc}\Big) = \mathsf{Constraints}\Big(\Gamma \vdash^{b}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}\Big)$$

$$\mathsf{Constraints}\Big(\Gamma_x \vdash^{b}_{\mathrm{FO}'\leq} (\nu x : \sigma^{c})\mathrm{P} : \mathsf{Proc}\Big) = \mathsf{Constraints}\Big(\Gamma_x, x : \sigma^{c} \vdash^{b}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}\Big)$$

$$\begin{aligned}\mathsf{Constraints}\Big(\Gamma, \Theta \vdash^{bc}_{\mathrm{FO}'\leq} \mathrm{P}|\mathrm{Q} : \mathsf{Proc}\Big) = \\ \mathsf{Constraints}\Big(\Gamma \vdash^{b}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}\Big) \cup \\ \mathsf{Constraints}\Big(\Theta \vdash^{c}_{\mathrm{FO}'\leq} \mathrm{Q} : \mathsf{Proc}\Big)\end{aligned}$$

$$\begin{aligned}\mathsf{Constraints}\Big(\Gamma, \Theta \vdash^{bc}_{\mathrm{FO}'\leq} \mathrm{P} + \mathrm{Q} : \mathsf{Proc}\Big) = \\ \mathsf{Constraints}\Big(\Gamma \vdash^{b}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}\Big) \cup \\ \mathsf{Constraints}\Big(\Theta \vdash^{c}_{\mathrm{FO}'\leq} \mathrm{Q} : \mathsf{Proc}\Big)\end{aligned}$$

$$\begin{aligned}\mathsf{Constraints}\Big(\Gamma_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^{b}})^{c} \vdash^{d}_{\mathrm{FO}'\leq} x(\overrightarrow{y} : \overrightarrow{\sigma^{b'}}).\mathrm{P} : \mathsf{Proc}\Big) = \\ \{\overrightarrow{b'} \leq_{\mathsf{B}} \overrightarrow{b}, d \leq_{\mathsf{B}} d', d \leq_{\mathsf{B}} c\} \cup \\ \mathsf{Constraints}\Big(\Gamma_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^{b}})^{c}, \overrightarrow{y} : \overrightarrow{\sigma^{b'}} \vdash^{d'}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}\Big)\end{aligned}$$

$$\begin{aligned}\mathsf{Constraints}\Big(\Gamma, x : (\overrightarrow{\sigma^{b}})^{c}, \overrightarrow{y} : \overrightarrow{\sigma^{b'}} \vdash^{d}_{\mathrm{FO}'\leq} \overline{x}[\overrightarrow{y}].\mathrm{P} : \mathsf{Proc}\Big) = \\ \{\overrightarrow{b} \leq_{\mathsf{B}} \overrightarrow{b'}, d \leq_{\mathsf{B}} d', d \leq_{\mathsf{B}} c\} \cup \\ \mathsf{Constraints}\Big(\Gamma, x : (\overrightarrow{\sigma^{b}})^{c}, \overrightarrow{y} : \overrightarrow{\sigma^{b'}} \vdash^{d'}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}\Big)\end{aligned}$$

$$\begin{aligned}\mathsf{Constraints}\Big(\Gamma_y, \Theta_y, x : (\sigma^{i})^{b} \vdash^{ic+\bar{i}d}_{\mathrm{FO}'\leq} x(y)_{\underset{\mathrm{certify}}{?}}\mathrm{P} \oplus \mathrm{Q} : \mathsf{Proc}\Big) = \\ \{c \leq_{\mathsf{B}} c', c \leq_{\mathsf{B}} b, d \leq_{\mathsf{B}} d', d \leq_{\mathsf{B}} b\} \\ \mathsf{Constraints}\Big(\Gamma_y, x : (\sigma^{i})^{b}, y : \sigma^{\mathrm{T}} \vdash^{c'}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}\Big) \cup \\ \mathsf{Constraints}\Big(\Gamma_y, x : (\sigma^{i})^{b}, y : \sigma^{\mathrm{U}} \vdash^{d'}_{\mathrm{FO}'\leq} \mathrm{Q} : \mathsf{Proc}\Big)\end{aligned}$$

Figure 5.4: Collecting Constraints for System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$

- $\mathrm{P} \equiv \mathrm{Q}|\mathrm{R}$: By the induction hypothesis, the result holds for the antecedents Q and R individually; that is

$$\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(Q) = \langle \Gamma, b, \mathcal{C}_1, \mathbb{S}_1 \rangle \Longrightarrow \Gamma \vdash^{b}_{\mathrm{FO}'\leq} \mathbb{S}_1(\mathrm{Q}) : \mathsf{Proc} \ \wedge \ \mathsf{Constraints}\big(\Gamma \vdash^{b}_{\mathrm{FO}'\leq} \mathbb{S}_1(\mathrm{Q}) : \mathsf{Proc}\big) = \mathcal{C}_1$$

and

$$\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathbb{S}_1(\mathrm{R})) = \langle \Theta, c, \mathcal{C}_2, \mathbb{S}_2 \rangle \Longrightarrow \Theta \vdash^{c}_{\mathrm{FO}'\leq} \mathbb{S}_1;\mathbb{S}_2(\mathrm{R}) : \mathsf{Proc} \ \wedge \ \mathsf{Constraints}\big(\Theta \vdash^{c}_{\mathrm{FO}'\leq} \mathbb{S}_1;\mathbb{S}_2(\mathrm{R}) : \mathsf{Proc}\big) = \mathcal{C}_2$$

  By the soundness of Unify, $\mathbb{S}_3 = \mathbb{S}_2;\mathsf{Unify}(\mathbb{S}_2(\Gamma), \Theta)$ implies $\mathbb{S}_3(\Gamma) \asymp \mathbb{S}_3(\Theta)$, hence by the composition rule

$$\mathbb{S}_3(\Gamma, \Theta \vdash^{bc}_{\mathrm{FO}'\leq} \mathrm{P}|\mathrm{Q} : \mathsf{Proc})$$

  and by definition

$$\mathsf{Constraints}\big(\mathbb{S}_3(\Gamma, \Theta \vdash^{bc}_{\mathrm{FO}'\leq} \mathrm{P}|\mathrm{Q} : \mathsf{Proc}\big) = \mathbb{S}_3(\mathcal{C}_1, \mathcal{C}_2)$$

  as required.

- $\mathrm{P} \equiv \mathrm{Q} + \mathrm{R}$: Similarly.

- $\mathrm{P} \equiv x(\overrightarrow{y} : \overrightarrow{\sigma^b}).\mathrm{Q}$: By the induction hypothesis the result holds for the antecedent individually; that is

$$\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathrm{Q}) = \langle \Gamma, c, \mathcal{C}, \mathbb{S}_1 \rangle \Longrightarrow \Gamma \vdash^{c}_{\mathrm{FO}'\leq} \mathbb{S}_1(\mathrm{Q}) : \mathsf{Proc} \ \wedge \ \mathsf{Constraints}\big(\Gamma \vdash^{c}_{\mathrm{FO}'\leq} \mathbb{S}_1(\mathrm{Q}) : \mathsf{Proc}\big) = \mathcal{C}$$

  Then by soundness of Unify, $\mathbb{S}_2 = \mathbb{S}_1;\mathsf{Unify}\Big(\Gamma, \mathbb{S}_1(\{x : (\overrightarrow{\sigma^j})^i, \overrightarrow{y} : \overrightarrow{\sigma^b}\})\Big)$ implies $\mathbb{S}_2(\Gamma) \asymp \mathbb{S}_2(\{x : (\overrightarrow{\sigma^j})^i, \overrightarrow{y} : \overrightarrow{\sigma^b}\})$; note that the same base types is used for both the types of $\overrightarrow{y}$ and the argument types of x, but fresh annotations variables are used — these are used to impose the sub-typing constraint. Also, by the soundness of $\mathbb{M}$, $\mathbb{S}_3 = \mathbb{S}_2;\mathbb{M}\Big(\mathbb{S}_2(i), \mathbb{S}_2(\overrightarrow{\sigma^j})\Big)$ implies $\mathbb{S}_3(j) = \mathbb{S}_3(i \cdot j')$ for every annotation j in $\overrightarrow{\sigma^j}$ and some j'. Then by the input type rule,

$$\mathbb{S}_3(\Gamma_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^j})^i \vdash^{k}_{(\mathrm{FO}'\leq)} x(\overrightarrow{y} : \overrightarrow{\sigma^b}).\mathrm{Q} : \mathsf{Proc})$$

and by definition

$$\mathsf{Constraints}\Big(\mathbb{S}_3(\Gamma_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^j})^i \vdash^k_{(\mathrm{FO}'\leq)} x(\overrightarrow{y} : \overrightarrow{\sigma^b}).\mathrm{Q} : \mathsf{Proc})\Big) = \\ \mathbb{S}_3(\mathcal{C} \cup \{\overrightarrow{b} \leq_{\mathsf{B}} \overrightarrow{j}, k \leq_{\mathsf{B}} c, k \leq_{\mathsf{B}} i\})$$

by the input type rule and definition of Constraints as required, noting that the relaxation rule may be used before the input rule, hence the fresh variable and corresponding constraint.

- $\mathrm{P} \equiv \overline{x}[\overrightarrow{y}].\mathrm{Q}$: Similarly, using fresh variables for the types of the argument names $\overrightarrow{y}$.
- $\mathrm{P} \equiv x(y : \sigma^b)_{\mathrm{certify}}?\mathrm{Q} \oplus \mathrm{R}$: By the induction hypothesis the result holds for the antecedents Q and R individually; that is

$$\mathsf{Type}_{\mathcal{T}_{\mathrm{FO}'\pi\leq}}(\mathrm{Q}) = \langle \Gamma, c, \mathcal{C}_1, S_1\rangle \Longrightarrow \\ \Gamma \vdash^c_{\mathrm{FO}'\leq} \mathbb{S}_1(\mathrm{Q}) : \mathsf{Proc} \ \wedge \ \mathsf{Constraints}\big(\Gamma \vdash^c_{\mathrm{FO}'\leq} \mathbb{S}_1(\mathrm{Q}) : \mathsf{Proc}\big) = \mathcal{C}_1$$

and

$$\mathsf{Type}_{\mathcal{T}_{\mathrm{FO}'\pi\leq}}(\mathbb{S}_1(\mathrm{R})) = \langle \Theta, d, \mathcal{C}_2, \mathbb{S}_2\rangle \Longrightarrow \\ \Theta \vdash^d_{\mathrm{FO}'\leq} \mathbb{S}_1; \mathbb{S}_2(\mathrm{R}) : \mathsf{Proc} \ \wedge \\ \mathsf{Constraints}\big(\Theta \vdash^d_{\mathrm{FO}'\leq} \mathbb{S}_1; \mathbb{S}_2(\mathrm{R}) : \mathsf{Proc}\big) = \mathcal{C}_2$$

Then, by the soundness of Unify:

  - $\mathbb{S}_3 = \mathbb{S}_1; \mathbb{S}_2; \mathsf{Unify}\big(\mathbb{S}_2(\Gamma), \mathbb{S}_1; \mathbb{S}_2(\{x : (\sigma^b)^i, y : \sigma^{\mathrm{T}}\})\big)$ implies
$$\mathbb{S}_3(\Gamma) \asymp \mathbb{S}_3(\{x : (\sigma^b)^i, y : \sigma^{\mathrm{T}}\})$$
  - $\mathbb{S}_4 = \mathbb{S}_3; \mathsf{Unify}\big(\mathbb{S}_3(\Theta), \mathbb{S}_3(\{x : (\sigma^b)^i, y : \sigma^{\mathrm{U}}\})\big)$ implies
$$\mathbb{S}_4(\Theta) \asymp \mathbb{S}_4(\{x : (\sigma^b)^i, y : \alpha^{\mathrm{U}}\})$$
  - $\mathbb{S}_5 = \mathbb{S}_4; \mathsf{Unify}(\mathbb{S}_4(\Gamma_y), \mathbb{S}_4(\Theta_y))$ implies $\mathbb{S}_5(\Gamma_y) \asymp \mathbb{S}_5(\Theta_y)$ (noting that the unification is performed in the absence of y due to the conflicting types).

Soundness of $\mathbb{M}$ means that $\mathbb{S}_6 = \mathbb{S}_5; \mathbb{M}\big(\mathbb{S}_5(i), \mathbb{S}_5(\sigma^b)\big)$ implies $\mathbb{S}_6(b) = \mathbb{S}_6(i \cdot i')$ for some i'. Then

$$\mathbb{S}_6(\Gamma_y, \Theta_y, x : (\sigma^b)^i \vdash^{bj+\bar{b}k}_{(\mathrm{FO}'\leq)} x(y : \sigma^b)_{\mathrm{certify}}?\mathrm{Q} \oplus \mathrm{R} : \mathsf{Proc})$$

and by definition

$$\mathsf{Constraints}\Big(\mathbb{S}_6(\Gamma_y, \Theta_y, x : (\sigma^b)^i \vdash^{bj+\bar{b}k}_{(\mathrm{FO}'\leq)} x(y : \sigma^b)_{\mathsf{certify}}?\mathrm{Q} \oplus \mathrm{R} : \mathsf{Proc})\Big) = \mathbb{S}_6(\mathcal{C}_1, \mathcal{C}_2, j \leq_{\mathsf{B}} c, j \leq_{\mathsf{B}} i, k \leq_{\mathsf{B}} d, k \leq_{\mathsf{B}} i)$$

by the certify type rule and definition of Constraints as required, noting that the relaxation rule may be used on both antecedents prior to the certify type rule, hence the fresh variables and corresponding constraints.

□

Completeness

Theorem 5.4.3 (Completeness of $\mathcal{T}_{\mathsf{FO}'\pi\leq}$) *$\forall \Gamma', \mathcal{C}', \mathrm{P}, b'$ such that*

$$\Gamma' \vdash^{b'}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}$$

is a valid deduction in System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$, and

$$\mathsf{Constraints}\Big(\Gamma' \vdash^{b'}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}\Big) = \mathcal{C}'$$

then

$$\langle \Gamma, b, \mathcal{C}, \mathbb{S}\rangle = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathrm{P})$$

is defined, with both $\Gamma \leq \Gamma'$ and $\mathcal{C} \leq \mathcal{C}'$.

Proof. By cases on the structure of P, then by inference on the derivation of $\Gamma' \vdash^{b'}_{\mathrm{FO}'\leq} \mathrm{P} : \mathsf{Proc}$ and by examination of the algorithm.

- $\mathrm{P} \equiv \mathbf{0}$: The deduction $\Gamma' \vdash^{b}_{(\mathrm{FO}'\leq)} \mathbf{0} : \mathsf{Proc}$ must have ended in a use of the zero inference rule, followed by zero or more instances of the weaken and relaxation rules. By examination of the algorithm $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathbf{0}) = \langle \emptyset, i, \emptyset, \mathsf{Id}\rangle$, and since $\mathsf{Constraints}\Big(\Gamma' \vdash^{b}_{(\mathrm{FO}'\leq)} \mathbf{0} : \mathsf{Proc}\Big) = \emptyset$ for all Γ', by definition $\emptyset \leq \Gamma'$ and $\emptyset \leq \emptyset$ as required.

- $\mathrm{P} \equiv !\mathrm{Q}$: The deduction $\Gamma' \vdash^{b}_{\mathrm{FO}'\leq} !\mathrm{Q} : \mathsf{Proc}$ must have ended in zero or more uses of the weaken and relaxation rules, preceded by an instance of the replication inference rule, with antecedent $\Gamma'' \vdash^{b'}_{\mathrm{FO}'\leq} \mathrm{Q} : \mathsf{Proc}$ (where $\Gamma'' \subseteq \Gamma'$ and $b \leq_{\mathsf{B}} b'$). By examination of the algorithm, $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(!\mathrm{Q}) = \mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathrm{Q})$, and by the induction hypothesis if $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathrm{Q}) = \langle \Gamma, c, \mathcal{C}, \mathbb{S}\rangle$ then $\Gamma \leq \Gamma''$ and $\mathcal{C} \leq \mathcal{C}'$ where $\mathcal{C}' = \mathsf{Constraints}\Big(\Gamma'' \vdash^{b'}_{\mathrm{FO}'\leq} \mathrm{Q} : \mathsf{Proc}\Big)$, hence by the replication rule and definition of Constraints result holds for !Q as well.

- $\mathrm{P} \equiv (\nu x : \sigma^d)\mathrm{Q}$: The deduction $\Gamma'_x \vdash^b_{\mathrm{FO}'\leq} (\nu x : \sigma^d)\mathrm{Q} : \mathsf{Proc}$ must have ended in zero or more instances of the weaken and relaxation rules, preceded by a use of the restriction rule with antecedent $\Gamma''_x, x : \sigma^d \vdash^{b'}_{\mathrm{FO}'\leq} \mathrm{Q} : \mathsf{Proc}$ where $\Gamma''_x \subseteq \Gamma'_x$ and $b \leq_{\mathsf{B}} b'$. By the induction hypothesis and inspection of the algorithm, $\mathsf{Type}_{\mathcal{T}_{\mathrm{FO}'\pi\leq}}(\mathrm{Q}) = \langle \Gamma, c, \mathcal{C}, \mathbb{S}_1 \rangle$ implies $\Gamma \leq \Gamma'', x : \sigma^d$ and $\mathcal{C} \leq \mathcal{C}'$, where $\mathcal{C}' = \mathsf{Constraints}\Big(\Gamma''_x, x : \sigma^d \vdash^{b'}_{\mathrm{FO}'\leq} \mathrm{Q} : \mathsf{Proc}\Big)$. By completeness of Unify, $\mathbb{S}_2 = \mathbb{S}_1; \mathsf{Unify}\big(\Gamma, \mathbb{S}_1(\{x : \sigma^b\})\big)$ is defined and complete. Hence $\mathbb{S}_2(\Gamma_x) \leq \Gamma'_x$ and $\mathbb{S}_2(\mathcal{C}) \leq \mathcal{C}'$ by the restriction rule and definition of $\mathsf{Constraints}$ as required.

- $\mathrm{P} \equiv \mathrm{Q}|\mathrm{R}$: The deduction $\Gamma', \Theta', \Delta \vdash^{b'}_{\mathrm{FO}'\leq} \mathrm{Q}|\mathrm{R} : \mathsf{Proc}$ must have ended in zero or more uses of the relaxation and weaken rules, preceded by a use of the composition rule with antecedents $\Gamma' \vdash^{b_1}_{\mathrm{FO}'\leq} \mathrm{Q} : \mathsf{Proc}$ and $\Theta' \vdash^{b_2}_{\mathrm{FO}'\leq} \mathrm{R} : \mathsf{Proc}$ where the domain of Δ contains all names introduced by weakening, and $b_1 \cdot b_2 = b \leq_{\mathsf{B}} b'$. By the induction hypothesis completeness holds for both individually; that is

$$\mathsf{Type}_{\mathcal{T}_{\mathrm{FO}'\pi\leq}}(\mathrm{Q}) = \langle \Gamma, c_1, \mathcal{C}_1, \mathbb{S}_1 \rangle \implies \Gamma \leq \Gamma' \wedge \mathcal{C}_1 \leq \mathcal{C}'_1$$

  given $\mathcal{C}'_1 = \mathsf{Constraints}\Big(\Gamma' \vdash^b_{\mathrm{FO}'\leq} \mathrm{Q} : \mathsf{Proc}\Big)$, and

$$\mathsf{Type}_{\mathcal{T}_{\mathrm{FO}'\pi\leq}}(\mathbb{S}_1(\mathrm{R})) = \langle \Theta, c, \mathcal{C}_2, \mathbb{S}_2 \rangle \implies \Theta \leq \Theta' \wedge \mathcal{C}_2 \leq \mathcal{C}'_2$$

  given $\mathcal{C}'_2 = \mathsf{Constraints}\Big(\Theta' \vdash^b_{\mathrm{FO}'\leq} \mathrm{R} : \mathsf{Proc}\Big)$. By completeness of Unify, since $\Gamma \leq \Gamma'$ and $\Theta \leq \Theta'$ (and $\Gamma' \asymp \Theta'$), then $\mathbb{S}_3 = \mathbb{S}_2; \mathsf{Unify}(\mathbb{S}_2(\Gamma), \Theta)$ is defined and most general. Hence by the composition rule and the definitions of $\mathsf{Type}_{\mathcal{T}_{\mathrm{FO}'\pi\leq}}$ and $\mathsf{Constraints}$, $\mathbb{S}_3(\Gamma, \Theta) \leq \Gamma', \Theta'$ and $\mathbb{S}_3(\mathcal{C}_1, \mathcal{C}_2) \leq \mathcal{C}'_1, \mathcal{C}'_2$ as required.

- $\mathrm{P} \equiv \mathrm{Q} + \mathrm{R}$: Similarly.

- $\mathrm{P} \equiv x(\overrightarrow{y} : \overrightarrow{\sigma^{b'}}).\mathrm{Q}$: The deduction

$$\Gamma'_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^b})^c, \Delta \vdash^d_{(\mathrm{FO}'\leq)} x(\overrightarrow{y} : \overrightarrow{\sigma^{b'}}).\mathrm{Q} : \mathsf{Proc}$$

  must have ended in zero or more uses of the weaken and relaxation rules, preceded by a use of the input rule with antecedent

$$\Gamma'_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma^{b'}} \vdash^{d'}_{\mathrm{FO}'\leq} \mathrm{Q} : \mathsf{Proc}$$

where $d \leq_{\mathsf{B}} d'$, $\overrightarrow{\sigma^b} \leq \overrightarrow{\sigma^{b'}}$, and the domain of Δ is all the names introduced via the final uses of the weaken rule. By the induction hypothesis $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathrm{Q}) = \langle \Gamma, e, \mathcal{C}, \mathbb{S}_1 \rangle$ is defined, with both $\Gamma \leq \Gamma'_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma^{b'}}$ and $\mathcal{C} \leq \mathcal{C}'$ given

$$\mathsf{Constraints}\Big(\Gamma'_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma^{b'}} \vdash^b_{\mathrm{FO}'\leq} \mathrm{Q} : \mathsf{Proc}\Big) = \mathcal{C}'$$

By the completeness of Unify, $\mathbb{S}_2 = \mathbb{S}_1; \mathsf{Unify}\Big(\Gamma, \mathbb{S}_1(\{x : (\overrightarrow{\sigma^j})^i, \overrightarrow{y} : \overrightarrow{\sigma^b}\})\Big)$ is both defined and most general (given $\Gamma \leq \Gamma', x : (\overrightarrow{\sigma^b})^c, \overrightarrow{y} : \overrightarrow{\sigma^{b'}}$ as before). Similarly, by completeness of $\mathbb{M}$, $\mathbb{S}_3 = \mathbb{S}_2; \mathbb{M}\Big(\mathbb{S}_2(i), \mathbb{S}_2(\overrightarrow{\sigma^j})\Big)$ is also defined and most general. Hence by the input rule and definition of $\mathsf{Constraints}$,

$$\mathbb{S}_3(\Gamma_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^j})^i) \leq \Gamma'_{\overrightarrow{y}}, x : (c \cdot \overrightarrow{\sigma^b})^c$$

and

$$\mathbb{S}_3(\mathcal{C} \cup \{\overrightarrow{b} \leq_{\mathsf{B}} \overrightarrow{j}, k \leq_{\mathsf{B}} c, k \leq_{\mathsf{B}} i\}) \leq \big(\mathcal{C}' \cup \{\overrightarrow{b'} \leq_{\mathsf{B}} \overrightarrow{b}, d \leq_{\mathsf{B}} d', d \leq_{\mathsf{B}} c\}\big)$$

as required.

- $\mathrm{P} \equiv \overline{x}[\overrightarrow{y}].\mathrm{Q}$: Similarly, with the algorithm using a fresh type variables for $\overrightarrow{y}$, and the sub-typing relation between the types carried by x and those of $\overrightarrow{y}$ (and thus the constraints generated) the reverse of the input case as required.

- $\mathrm{P} \equiv x(y : \sigma^i)_{\mathrm{certify}}?\mathrm{Q} \oplus \mathrm{R}$: The deduction

$$\Gamma', \Theta', \Delta x : (\sigma^i)^b \vdash^{ic+\bar{i}d}_{(\mathrm{FO}'\leq)} x(y : \sigma^i)_{\mathrm{certify}}?\mathrm{Q} \oplus \mathrm{R} : \mathsf{Proc}$$

must have ended in zero or more uses of the weaken and relaxation rules, preceded by a use of the certify rule with antecedents

$$\begin{aligned} \Gamma', x : (\sigma^i)^b, y : \sigma^{\mathrm{T}} \ &\vdash^{c'}_{\mathrm{FO}'\leq} \ \mathrm{Q} : \mathsf{Proc} \\ \text{and} \quad \Theta', x : (\sigma^i)^b, y : \sigma^{\mathrm{U}} \ &\vdash^{d'}_{\mathrm{FO}'\leq} \ \mathrm{R} : \mathsf{Proc} \end{aligned}$$

where the domain of Δ is all the free names introduced via the weakening rule, $d \leq_{\mathsf{B}} d'$ and $c \leq_{\mathsf{B}} c'$, and assuming for notational convenience

that $y \notin \mathrm{dom}\Gamma' \cup \Theta'$. By the induction hypothesis completeness holds for the antecedents individually; that is

$$\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathrm{Q}) = \langle \Gamma, e, \mathcal{C}_1, \mathbb{S}_1 \rangle \implies \Gamma \leq \Gamma', x : (\sigma^i)^b, y : \sigma^{\mathrm{T}} \ \wedge \ \mathcal{C}_1 \leq \mathcal{C}_1'$$

where $\mathcal{C}_1' = \mathsf{Constraints}\Big(\Gamma', x : (\sigma^i)^b, y : \sigma^{\mathrm{T}} \vdash^{c'}_{\mathrm{FO}'\leq} \mathrm{Q} : \mathsf{Proc}\Big)$, and

$$\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathbb{S}_1(\mathrm{R})) = \langle \Theta, f, \mathcal{C}_2, \mathbb{S}_2 \rangle \implies \Theta \leq \Theta', x : (\sigma^i)^b, y : \sigma^{\mathrm{U}} \ \wedge \ \mathcal{C}_2 \leq \mathcal{C}_2'$$

where $\mathcal{C}_2' = \mathsf{Constraints}\Big(\Theta', x : (\sigma^i)^b, y : \sigma^{\mathrm{U}} \vdash^{d'}_{\mathrm{FO}'\leq} \mathrm{R} : \mathsf{Proc}\Big)$. By the completeness of Unify the following are all defined and most general:

- $\mathbb{S}_3 = \mathbb{S}_1; \mathbb{S}_2; \mathsf{Unify}\big(\mathbb{S}_2(\Gamma), \mathbb{S}_1; \mathbb{S}_2(\{x : (\sigma^i)^j, y : \sigma^{\mathrm{T}}\})\big)$;
- $\mathbb{S}_4 = \mathbb{S}_3; \mathsf{Unify}\big(\mathbb{S}_3(\Theta), \mathbb{S}_3(\{x : (\sigma^i)^j, y : \sigma^{\mathrm{U}}\})\big)$; and
- $\mathbb{S}_5 = \mathbb{S}_4; \mathsf{Unify}\big(\mathbb{S}_4(\Gamma_y), \mathbb{S}_4(\Theta_y)\big)$;

Similarly, by completeness of $\mathbb{M}$, $\mathbb{S}_6 = \mathbb{S}_5; \mathbb{M}(\mathbb{S}_5(j), \mathbb{S}_5(\sigma^i))$ is also defined and most general. Hence by the definition of $\mathsf{Constraints}$ and the algorithm, the algorithm is defined, with

$$\mathbb{S}_6(\Gamma_y, \Theta_y, x : (\sigma^b)^i) \leq \Gamma', \Theta', x : (\sigma^i)^b$$

and

$$\mathbb{S}_6(\mathcal{C}_1, \mathcal{C}_2, j \leq_{\mathsf{B}} e, j \leq_{\mathsf{B}} i, k \leq_{\mathsf{B}} f, k \leq_{\mathsf{B}} i) \leq \mathcal{C}_1', \mathcal{C}_2', d \leq_{\mathsf{B}} d', c \leq_{\mathsf{B}} c', c \leq_{\mathsf{B}} b, d \leq_{\mathsf{B}} b$$

as required.

□

5.4.3 Termination of $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}$

As for the algorithm $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}\pi}}$ (see Theorem 5.3.3), it may be shown that the algorithm $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}$ also terminates.

Theorem 5.4.4 (Termination of $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}$) *The algorithm $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}$ terminates for all finite inputs.*

Proof. By induction on the structure of the input. Each step either: terminates immediately (returning a result); calls one of the algorithms Unify or $\mathbb{M}$ which terminates by Theorems 5.2.15 and 5.2.21; or makes a recursive call on a subset of the input. Since each stage either terminates explicitly or recurses on a subset of the input, if the input is finite then the algorithm must eventually terminate.

As an example, consider the case of $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\overline{x}[\overrightarrow{y}].\mathrm{P})$: there are three steps involved, all of which can be shown to terminate. Firstly, since according to the statement the input is finite, then a subset of the input is also finite and thus $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\mathrm{P})$ terminates by the induction hypothesis. Secondly, by the statement the sequence $\overrightarrow{y}$ must be finite so by Theorem 5.2.15 the call to $\mathsf{Unify}\Big(\Gamma, \{x : (\overrightarrow{\alpha^j})^i, \overrightarrow{y} : \overrightarrow{\alpha^k}\}\Big)$ terminates. Lastly, the call to $\mathbb{M}\Big(\mathbb{S}_2(i), \mathbb{S}_2(\overrightarrow{\alpha^j})\Big)$ terminates by Theorem 5.2.21. Hence, since all sub-steps terminate, the algorithm for $\mathsf{Type}_{\mathcal{T}_{\mathsf{FO}'\pi\leq}}(\overline{x}[\overrightarrow{y}].\mathrm{P})$ must also terminate. □

5.5 Type Inference For System $\mathcal{T}_{\mathsf{HO}\pi\leq}$

Type inference for system $\mathcal{T}_{\mathsf{HO}\pi\leq}$ is only slightly more problematic than the previous two systems. It is the treatment of sub-typing that requires the most care, due to the current definition of sub-typing which only applies to the outer-most annotations on channel types.

This situation is handled with the definition of an auxiliary function (Definition 5.5.1) that takes a set of type constraints, and returns the corresponding set of annotation constraints.

Definition 5.5.1 *The function $[\![\cdot]\!]$ returns the set of annotation constraints corresponding to the argument set of sub-type constraints. By definition, the number of constraints returns is less-than or equal to the number of sub-type constraints in the argument sequence.*

$$\begin{array}{rcl} [\![]\!] & \triangleq & \emptyset \\ [\![(\overrightarrow{\sigma^c})^b \leq (\overrightarrow{\sigma^c})^{b'}]\!] & \triangleq & \{b' \leq_{\mathsf{B}} b\} \\ [\![\sigma^b\overrightarrow{\sigma^c} \leq \sigma^{b'}\overrightarrow{\sigma^{c'}}]\!] & \triangleq & [\![\sigma^b \leq \sigma^{b'}]\!] \cup [\![\overrightarrow{\sigma^c} \leq \overrightarrow{\sigma^{c'}}]\!] \\ [\![\_ \leq \_]\!] & \triangleq & \emptyset \end{array}$$

A companion function takes a similar set of constraints, and returns an annotation unifier for those (process and abstraction) types that require matching annotations (Definition 5.5.2).

Definition 5.5.2 *The function $\{\![\cdot]\!\}$ takes a set of sub-type constraints, and returns an annotation unifier which unifies all top-level annotations on process and abstraction types.*

$$\begin{aligned}
\{\![]\!\} &\triangleq \mathsf{Id} \\
\{\![\mathsf{Proc}^b \leq \mathsf{Proc}^c]\!\} &\triangleq \mathsf{BUNIFY}(b, c) \\
\{\![(\overrightarrow{\sigma_1^d}) \to \mathsf{Proc}^b \leq (\overrightarrow{\sigma_2^e}) \to \mathsf{Proc}^c]\!\} &\triangleq \mathsf{BUNIFY}(b, c) \\
\{\![\sigma^b \overrightarrow{\sigma^c} \leq \sigma^{b'} \overrightarrow{\sigma^{c'}}]\!\} &\triangleq \mathit{let}\ \mathbb{R} = \{\![\sigma^b \leq \sigma^{b'}]\!\} \\
&\quad\ \mathit{in}\ \mathbb{R}; \{\![\mathbb{R}(\overrightarrow{\sigma^c} \leq \overrightarrow{\sigma^{c'}})]\!\} \\
\{\![_ \leq _]\!\} &\triangleq \emptyset
\end{aligned}$$

The two functions are separate as both are used in the implementation, but only the former is of use in the definition of constraints in a deduction.

5.5.1 Implementation

The algorithm proceeds in essentially the same manner as before; the main difference is the additional constraints mentioned above. Note the need to also carry $\mathbb{C}_\epsilon$ throughout the inference process, although it is written as part of the algorithm rather than an argument as it is constant throughout the entire deduction. The algorithm itself is presented across Figures 5.5, 5.6, 5.7, and 5.8. It takes as input a valid term in $\mathcal{T}_{\mathsf{HO}\pi\leq}$, and returns a tuple consisting of a type environment, a set of constraints (see Definition 5.4.1), the type assigned to the term (note that this is now *not* restricted to just the well-formed process type Proc due to the annotation on the type and the possibility of abstractions), the contextual information ($\mathbb{C}$) of the term, and a substitution. An auxiliary algorithm (Figure 5.9) is also needed to handle the sequences of terms encountered in the output and application constructs, as they may contain both names *and* processes.

5.5.2 Correctness of $\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}$

Soundness

First some auxiliary definitions and results are required.

The constraints generated by a deduction in $\mathcal{T}_{\mathsf{HO}\pi\leq}$ are defined in a similar manner to system $\mathcal{T}_{\mathsf{FO}\pi\leq}$ by the algorithm Constraints; shown in Figures 5.10 and 5.11. As with system $\mathcal{T}_{\mathsf{FO}\pi\leq}$, note that the definition is over a deduction tree (in which all antecedents are known) not a term in isolation.

$$
\begin{array}{rcl}
\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathbf{0}^b) & = & \langle \emptyset, \emptyset, \mathsf{Proc}^b, \emptyset, \mathsf{Id} \rangle \\
\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(X) & = & \langle \{X : \alpha^i\}, \emptyset, \alpha^i, \{X : i\}, \mathsf{Id} \rangle \\
\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(!\mathrm{P}) & = & \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{P}) \\
\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}((\nu x : \sigma^c)\mathrm{P}) & = & \text{let } \langle \Gamma, \mathcal{C}, \mathsf{Proc}^b, \mathbb{C}, \mathbb{S}_1 \rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{P}) \\
 & & \quad \mathbb{S}_2 = \mathbb{S}_1; \mathsf{Unify}(\Gamma, \mathbb{S}_1(\{x : \sigma^c\})) \\
 & & \quad \text{in } \langle \mathbb{S}_2(\Gamma_x), \mathbb{S}_2(\mathcal{C}), \mathbb{S}_2(\mathsf{Proc}^b), \mathbb{S}_2(\mathbb{C}_x), \mathbb{S}_2 \rangle \\
\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}\Big((\overrightarrow{V} : \overrightarrow{\sigma^c})\mathrm{P}\Big) & = & \text{let } \langle \Gamma, \mathcal{C}, \mathsf{Proc}^b, \mathbb{C}, \mathbb{S}_1 \rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{P}) \\
 & & \quad \mathbb{S}_2 = \mathbb{S}_1; \mathsf{Unify}\Big(\Gamma, \mathbb{S}_1(\{\overrightarrow{V} : \overrightarrow{\sigma^c}\})\Big) \\
 & & \quad \text{in } \Big\langle \mathbb{S}_2(\Gamma_{\overrightarrow{V}}), \mathbb{S}_2(\mathcal{C}, \forall V \in \overrightarrow{V}.b \leq_{\mathsf{B}} \mathbb{C}_{\mathrm{T}}(V)), \\
 & & \qquad \mathbb{S}((\overrightarrow{\sigma^c}) \to \mathsf{Proc}^b), \mathbb{S}_2(\mathbb{C}_{\overrightarrow{V}}), \mathbb{S}_2 \Big\rangle \\
\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}\Big(\mathrm{A}\langle\overrightarrow{K}\rangle\Big) & = & \text{let } \langle \Gamma, \mathcal{C}_1, \sigma^c, \mathbb{C}_1, \mathbb{S}_1 \rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{A}) \\
 & & \quad \Big\langle \overrightarrow{\Theta}, \mathcal{C}_2, \overrightarrow{\sigma'^d}, \overrightarrow{\mathbb{C}_2}, \mathbb{S}_2 \Big\rangle = \mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}\Big(\mathbb{S}_1(\overrightarrow{K})\Big) \\
 & & \quad \mathbb{S}_3 = \mathbb{S}_1; \mathbb{S}_2; \mathsf{Unify}\Big(\Gamma, \mathbb{S}_1; \mathbb{S}_2(\bigcup \overrightarrow{\Theta})\Big) \\
 & & \quad \mathbb{S}_4 = \mathbb{S}_3; \mathcal{U}\Big(\mathbb{S}_3(\sigma^c), (\overrightarrow{\alpha^i}) \to \mathsf{Proc}^{\mathbb{S}_3(b)}\Big) \\
 & & \quad \mathbb{S}_5 = \mathbb{S}_4; \mathsf{CUnify}\Big(\mathbb{S}_4(\mathbb{C}_1), \mathbb{S}_4(\bigcup \overrightarrow{\mathbb{C}_2}, \mathsf{chans}\Big(\overrightarrow{K}\Big) : c)\Big) \\
 & & \quad \mathbb{S}_6 = \mathbb{S}_5; \mathcal{U}\Big(\mathbb{S}_5(\overrightarrow{\alpha^b}), \mathbb{S}_5(\overrightarrow{\sigma'^b})\Big) \\
 & & \quad \mathbb{S}_7 = \mathbb{S}_6; \{\![\mathbb{S}_6(\overrightarrow{\sigma'^d} \leq \overrightarrow{\alpha^i})]\!\} \\
 & & \quad \text{in } \Big\langle \mathbb{S}_7(\Gamma, \overrightarrow{\Theta}), \mathbb{S}_7(\mathcal{C}_1, \mathcal{C}_2, [\![\overrightarrow{\sigma'^d} \leq \overrightarrow{\alpha^i}]\!], \\
 & & \qquad \forall x \in \mathbb{C}_\epsilon.c \leq_{\mathsf{B}} \overrightarrow{\mathbb{C}_{2\mathrm{T}}}(x), \\
 & & \qquad \forall \mathbb{C}_{2j}, V \in \mathrm{dom}\mathbb{C}_{2j}.\mathbb{C}_{2j} \leq_{\mathsf{B}} d_j), \\
 & & \qquad \mathbb{S}_7(\mathsf{Proc}^c), \mathbb{S}_7(\mathbb{C}_1, \overrightarrow{\mathbb{C}_2}, \mathsf{chans}\Big(\overrightarrow{K}\Big) : c), \mathbb{S}_7 \Big\rangle
\end{array}
$$

Figure 5.5: Type Inference Algorithm for System $\mathcal{T}_{\mathsf{HO}\pi\leq}$

$$
\begin{array}{rcl}
\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{P}|\mathrm{Q}) & = & \text{let } \left\langle \Gamma, \mathcal{C}_1, \mathsf{Proc}^b, \mathbb{C}_1, \mathbb{S}_1 \right\rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{P}) \\
& & \quad \langle \Theta, \mathcal{C}_2, \mathsf{Proc}^c, \mathbb{C}_2, \mathbb{S}_2 \rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathbb{S}_1(\mathrm{Q})) \\
& & \quad \mathbb{S}_3 = \mathbb{S}_1; \mathbb{S}_2; \mathsf{Unify}(\mathbb{S}_2(\Gamma), \Theta) \\
& & \quad \mathbb{S}_4 = \mathbb{S}_3; \mathsf{CUnify}(\mathbb{S}_3(\mathbb{C}_1), \mathbb{S}_3(\mathbb{C}_2)) \\
& & \text{in } \left\langle \mathbb{S}_4(\Gamma, \Theta), \mathbb{S}_4(\mathcal{C}_1, \mathcal{C}_2), \mathsf{Proc}^{\mathbb{S}_4(bc)}, \mathbb{S}_4(\mathbb{C}_1, \mathbb{C}_2), \mathbb{S}_4 \right\rangle \\
\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{P}+\mathrm{Q}) & = & \text{let } \left\langle \Gamma, \mathcal{C}_1, \mathsf{Proc}^b, \mathbb{C}_1, \mathbb{S}_1 \right\rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{P}) \\
& & \quad \langle \Theta, \mathcal{C}_2, \mathsf{Proc}^c, \mathbb{C}_2, \mathbb{S}_2 \rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathbb{S}_1(\mathrm{Q})) \\
& & \quad \mathbb{S}_3 = \mathbb{S}_1; \mathbb{S}_2; \mathsf{Unify}(\mathbb{S}_2(\Gamma), \Theta) \\
& & \quad \mathbb{S}_4 = \mathbb{S}_3; \mathsf{CUnify}(\mathbb{S}_3(\mathbb{C}_1), \mathbb{S}_3(\mathbb{C}_2)) \\
& & \text{in } \left\langle \mathbb{S}_4(\Gamma, \Theta), \mathbb{S}_4(\mathcal{C}_1, \mathcal{C}_2), \mathsf{Proc}^{\mathbb{S}_4(bc)}, \mathbb{S}_4(\mathbb{C}_1, \mathbb{C}_2), \mathbb{S}_4 \right\rangle \\
\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}\Big(x(\overrightarrow{V} : \overrightarrow{\sigma^c}).\mathrm{P}\Big) & = & \text{let } \left\langle \Gamma, \mathcal{C}, \mathsf{Proc}^b, \mathbb{C}, \mathbb{S}_1 \right\rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{P}) \\
& & \quad \mathbb{S}_2 = \mathbb{S}_1; \mathsf{Unify}\Big(\Gamma, \mathbb{S}_1(\{x : (\overrightarrow{\sigma^j})^i, \overrightarrow{V} : \overrightarrow{\sigma^c}\})\Big) \\
& & \quad \mathbb{S}_3 = \mathbb{S}_2; \mathbb{M}\Big(\mathbb{S}_2(i), \mathbb{S}_2(\overrightarrow{\sigma^j})\Big) \\
& & \quad \mathbb{S}_4 = \mathbb{S}_3; \mathsf{CUnify}(\mathbb{S}_3(\mathbb{C}), \mathbb{S}_3(\{x : b\})) \\
& & \quad \mathbb{S}_5 = \mathbb{S}_4; \{[\![\mathbb{S}_4(\overrightarrow{\sigma^j} \leq \overrightarrow{\sigma^c})]\!]\} \\
& & \text{in } \Big\langle \mathbb{S}_5(\Gamma_{\overrightarrow{V}}, x : (\overrightarrow{\sigma^j})^i), \mathbb{S}_5(\mathcal{C}, [\![\overrightarrow{\sigma^j} \leq \overrightarrow{\sigma^c}]\!], \\
& & \qquad \forall V \in \overrightarrow{V}.b \leq_{\mathsf{B}} \mathbb{C}_{\mathrm{T}}(V)), \mathbb{S}_5(\mathsf{Proc}^b), \mathbb{S}_5(\mathbb{C}_{\overrightarrow{V}}, x : b) \Big\rangle
\end{array}
$$

Figure 5.6: Type Inference Algorithm for System $\mathcal{T}_{\mathsf{HO}\pi\leq}$ (cont.)

$$
\begin{aligned}
\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}\Big(\overline{x}[\overrightarrow{K}].\mathrm{P}\Big) \;=\; & \text{let } \big\langle \Gamma, \mathcal{C}_1, \mathsf{Proc}^b, \mathbb{C}_1, \mathbb{S}_1 \big\rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{P}) \\
& \Big\langle \overrightarrow{\Theta}, \mathcal{C}_2, \overrightarrow{\sigma^c}, \overrightarrow{\mathbb{C}_2}, \mathbb{S}_2 \Big\rangle = \mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}\Big(\mathbb{S}_1(\overrightarrow{K})\Big) \\
& \mathbb{S}_3 = \mathbb{S}_1; \mathbb{S}_2; \mathsf{Unify}\Big(\Gamma, \mathbb{S}_1(\bigcup \overrightarrow{\Theta})\Big) \\
& \mathbb{S}_4 = \mathbb{S}_3; \mathsf{Unify}\Big(\mathbb{S}_3(\Gamma, \overrightarrow{\Theta}), \{x : (\overrightarrow{\sigma^j})^i\}\Big) \\
& \mathbb{S}_5 = \mathbb{S}_4; \mathbb{M}\Big(\mathbb{S}_4(b), \mathbb{S}_4(\overrightarrow{\sigma^j})\Big) \\
& \mathbb{S}_6 = \mathbb{S}_5; \mathbb{M}\Big(\mathbb{S}_3(i), \mathbb{S}_3(\overrightarrow{\sigma^j})\Big) \\
& \mathbb{S}_7 = \mathbb{S}_6; \mathsf{CUnify}\Big(\mathbb{S}_6(\mathbb{C}_1), \mathbb{S}_6(\bigcup \overrightarrow{\mathbb{C}_2})\Big) \\
& \mathbb{S}_8 = \mathbb{S}_7; \mathsf{CUnify}\Big(\mathbb{S}_7(\mathbb{C}_1, \overrightarrow{\mathbb{C}_2}), \mathbb{S}_7(\mathsf{chans}\Big(\overrightarrow{K}\Big) : b, x : b)\Big) \\
& \mathbb{S}_9 = \mathbb{S}_8; \{[\![\mathbb{S}_8(\overrightarrow{\sigma^c} \leq \overrightarrow{\sigma^j})]\!]\} \\
\text{in } & \Big\langle \mathbb{S}_9(\Gamma, \overrightarrow{\Theta}, x : (\overrightarrow{\sigma^j})^i), \\
& \quad \mathbb{S}_9(\mathcal{C}_1, \mathcal{C}_2, [\![\overrightarrow{\sigma^c} \leq \overrightarrow{\sigma^j}]\!], \forall x \in \mathbb{C}_\epsilon . b \leq_{\mathsf{B}} \overrightarrow{\mathbb{C}_{2\mathrm{T}}}(x), \\
& \qquad \forall \mathbb{C}_{2k}, V \in \mathrm{dom}\mathbb{C}_{2j} . \mathbb{C}_{2k}(V) \leq_{\mathsf{B}} c_k), \\
& \quad \mathbb{S}_9(\mathsf{Proc}^c), \mathbb{S}_9(\mathbb{C}_1, \overrightarrow{\mathbb{C}_2}, x : b, \mathsf{chans}\Big(\overrightarrow{K}\Big) : b)\Big\rangle
\end{aligned}
$$

Figure 5.7: Type Inference Algorithm for System $\mathcal{T}_{\mathsf{HO}\pi\leq}$ (cont.)

$$
\begin{aligned}
&\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}\big(x(V:\sigma^i)_{\mathrm{certify}}\mathrm{P}\oplus\mathrm{Q}\big) = \\
&\qquad \text{let } \big\langle \Gamma, \mathcal{C}_1, \mathsf{Proc}^b, \mathbb{C}_1, \mathbb{S}_1\big\rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{P}) \\
&\qquad\quad \langle \Theta, \mathcal{C}_2, \mathsf{Proc}^c, \mathbb{C}_2, \mathbb{S}_2\rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathbb{S}_1(\mathrm{Q})) \\
&\qquad\quad \mathbb{S}_3 = \mathbb{S}_1;\mathbb{S}_2;\mathsf{Unify}\big(\Gamma, \mathbb{S}_1;\mathbb{S}_2(\{x:(\sigma^i)^j, V:\sigma^{\mathrm{T}}\})\big) \\
&\qquad\quad \mathbb{S}_4 = \mathbb{S}_3;\mathsf{Unify}\big(\mathbb{S}_3(\Theta), \mathbb{S}_3(\{x:(\sigma^i)^j, V:\sigma^{\mathrm{U}}\})\big) \\
&\qquad\quad \mathbb{S}_5 = \mathbb{S}_4;\mathsf{Unify}\big(\mathbb{S}_4(\Gamma_V), \mathbb{S}_4(\Theta_V)\big) \\
&\qquad\quad \mathbb{S}_6 = \mathbb{S}_5;\mathbb{M}\big(\mathbb{S}_5(j), \mathbb{S}_5(\sigma^i)\big) \\
&\qquad\quad \mathbb{S}_7 = \mathbb{S}_6;\mathsf{CUnify}\big(\mathbb{S}_6(\mathbb{C}_{1V}\ \langle i\rangle\ \mathbb{C}_{2V}), \\
&\qquad\qquad\qquad\qquad \mathbb{S}_6(\{x: i\cdot b + \bar{i}\cdot c\})\big) \\
&\qquad \text{in } \Big\langle \mathbb{S}_7(\Gamma_V, \Theta_V, x:(\sigma^i)^j), \\
&\qquad\quad \mathbb{S}_7(\mathcal{C}_1, \mathcal{C}_2, b \leq_{\mathsf{B}} \mathbb{C}_{1\mathrm{T}}(V), c \leq_{\mathsf{B}} \mathbb{C}_{2\mathrm{T}}(V)), \\
&\qquad\quad \mathbb{S}_7(\mathsf{Proc}^{i\cdot b+\bar{i}\cdot c}), \\
&\qquad\quad \mathbb{S}_7(\mathbb{C}_{1V}\ \langle i\rangle\ \mathbb{C}_{2V}, x: i\cdot b+\bar{i}\cdot c), \mathbb{S}_7\Big\rangle
\end{aligned}
$$

Figure 5.8: Type Inference Algorithm for System $\mathcal{T}_{\mathsf{HO}\pi\leq}$ (cont.)

$$
\begin{aligned}
\mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(x) &= \big\langle \{x:\alpha^i\}, \emptyset, \alpha^i, \emptyset, \mathsf{Id}\big\rangle \\
\mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{A}) &= \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{A}) \\
\mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}\Big(K\overrightarrow{K'}\Big) &= \text{let } \big\langle \Gamma, \mathcal{C}, \sigma^b, \mathbb{C}, S_1\big\rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(K) \\
&\qquad \Big\langle \overrightarrow{\Gamma'}, \mathcal{C}', \overrightarrow{\sigma'^c}, \overrightarrow{\mathbb{C}'}, \mathbb{S}_2\Big\rangle = \mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}\Big(\overrightarrow{K'}\Big) \\
&\qquad \mathbb{S}_3 = \mathbb{S}_1;\mathbb{S}_2;\mathsf{Unify}\Big(\Gamma, \bigcup\overrightarrow{\Gamma'}\Big) \\
&\qquad \mathbb{S}_4 = \mathbb{S}_3;\mathsf{CUnify}\Big(\mathbb{S}_3(\mathbb{C}), \mathbb{S}_3(\bigcup\overrightarrow{\mathbb{C}'})\Big) \\
&\qquad \text{in } \Big\langle \mathbb{S}_4(\Gamma\overrightarrow{\Gamma'}), \mathbb{S}_4(\mathcal{C}, \mathcal{C}'), \mathbb{S}_4(\sigma^b\overrightarrow{\sigma'^d}), \mathbb{S}_4(\mathbb{C}\overrightarrow{\mathbb{C}'}), \mathbb{S}_4\Big\rangle
\end{aligned}
$$

Figure 5.9: Type Inference Algorithm for System $\mathcal{T}_{\mathsf{HO}\pi\leq}$ Sequences

$$\mathsf{Constraints}\big(\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbf{0} : \mathsf{Proc}^b; \emptyset\big) = \emptyset$$

$$\mathsf{Constraints}\big(\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} x : \sigma^b; \emptyset\big) = \emptyset$$

$$\mathsf{Constraints}\big(\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} X : \sigma^b; \mathbb{C}\big) = \emptyset$$

$$\mathsf{Constraints}\Big(\overrightarrow{\Gamma} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma^b}; \overrightarrow{\mathbb{C}}\Big) =$$
$$\mathsf{Constraints}\big(\Gamma_1 \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} K_1 : \sigma_1^{b_1}; \mathbb{C}_1\big) \cup \ldots \cup$$
$$\mathsf{Constraints}\big(\Gamma_n \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} K_n : \sigma_n^{b_n}; \mathbb{C}_n\big)$$

$$\mathsf{Constraints}\big(\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} !\mathrm{P} : \mathsf{Proc}^b; \mathbb{C}\big) = \mathsf{Constraints}\big(\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C}\big)$$

$$\mathsf{Constraints}\big(\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} (\nu x : \sigma^c)\mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_x\big) =$$
$$\mathsf{Constraints}\big(\Gamma_x, x : \sigma^b \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C}\big)$$

$$\mathsf{Constraints}\big(\Gamma, \Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P}|\mathrm{Q} : \mathsf{Proc}^b; \mathbb{C}_1, \mathbb{C}_2\big) =$$
$$\mathsf{Constraints}\big(\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{b_1}; \mathbb{C}_1\big) \cup$$
$$\mathsf{Constraints}\big(\Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{Q} : \mathsf{Proc}^{b_2}; \mathbb{C}_2\big)$$

$$\mathsf{Constraints}\big(\Gamma, \Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} + \mathrm{Q} : \mathsf{Proc}^{bc}; \mathbb{C}_1, \mathbb{C}_2\big) =$$
$$\mathsf{Constraints}\big(\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_1\big) \cup$$
$$\mathsf{Constraints}\big(\Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{Q} : \mathsf{Proc}^c; \mathbb{C}_2\big)$$

$$\mathsf{Constraints}\Big(\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} (\overrightarrow{V} : \overrightarrow{\sigma^c})\mathrm{P} : (\overrightarrow{\sigma^c}) \to \mathsf{Proc}^b; \mathbb{C}_{\overrightarrow{V}}\Big) =$$
$$\{\forall V \in \overrightarrow{V}.b \leq_{\mathsf{B}} \mathbb{C}_{\mathrm{T}}(V)\} \cup$$
$$\mathsf{Constraints}\Big(\Gamma_{\overrightarrow{V}}, \overrightarrow{V} : \overrightarrow{\sigma^c} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C}\Big)$$

Figure 5.10: Constraints Generated by a Deduction in $\mathcal{T}_{\mathsf{HO}\pi\leq}$

$$
\begin{aligned}
&\mathsf{Constraints}\Big(\Gamma, \overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{A}\langle\overrightarrow{K}\rangle : \mathsf{Proc}^b; \mathbb{C}_1, \overrightarrow{\mathbb{C}_2}, \mathsf{chans}\Big(\overrightarrow{K}\Big) : b\Big) = \\
&\qquad \{[\![\overrightarrow{\sigma^d} \leq \overrightarrow{\sigma^c}]\!], \forall x \in \mathbb{C}_\epsilon.b \leq_{\mathsf{B}} \overrightarrow{\mathbb{C}'_{2\mathrm{T}}}(x), \\
&\qquad\quad \forall \mathbb{C}_{2i}, V \in \mathrm{dom}\mathbb{C}_{2i}.\mathbb{C}_{2i}(V) \leq_{\mathsf{B}} d_i\} \cup \\
&\qquad \mathsf{Constraints}\Big(\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{A} : (\overrightarrow{\sigma^c}) \to \mathsf{Proc}^b; \mathbb{C}_1\Big) \cup \\
&\qquad \mathsf{Constraints}\Big(\overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma^d}; \overrightarrow{\mathbb{C}_2}\Big) \\
&\mathsf{Constraints}\Big(\Gamma, x : (\overrightarrow{\sigma^c})^b \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} x(\overrightarrow{V}).\mathrm{P} : \mathsf{Proc}^d; \mathbb{C}_{\overrightarrow{V}}, x : d\Big) = \\
&\qquad \{[\![\overrightarrow{\sigma^c} \leq \overrightarrow{\sigma^{c'}}]\!], \forall V \in \overrightarrow{V}.d' \leq_{\mathsf{B}} \mathbb{C}_{\mathrm{T}}(V)\} \cup \\
&\qquad \mathsf{Constraints}\Big(\Gamma_{\overrightarrow{V}}, x : (\overrightarrow{\sigma^c})^b, \overrightarrow{V} : \overrightarrow{\sigma^{c'}} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^d; \mathbb{C}\Big) \\
&\mathsf{Constraints}\Big(\Gamma, x : (c \cdot \overrightarrow{\sigma^d})^b, \overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overline{x}[\overrightarrow{K}].\mathrm{P} : \mathsf{Proc}^c; \\
&\qquad\qquad\qquad\qquad \mathbb{C}_1, x : c, \overrightarrow{\mathbb{C}_2}, \mathsf{chans}\Big(\overrightarrow{K}\Big) : c\Big) = \\
&\qquad \{[\![\overrightarrow{\sigma^{d'}} \leq c \cdot \overrightarrow{\sigma^d}]\!], \forall x \in \mathbb{C}_\epsilon.c \leq_{\mathsf{B}} \overrightarrow{\mathbb{C}'_{2\mathrm{T}}}(x), \\
&\qquad\quad \forall \mathbb{C}_{2i}, V \in \mathrm{dom}\mathbb{C}_{2i}.\mathbb{C}_{2i}(V) \leq_{\mathsf{B}} d'_i\} \cup \\
&\qquad \mathsf{Constraints}\Big(\Gamma, x : (c \cdot \overrightarrow{\sigma^d})^b \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^c; \mathbb{C}_1\Big) \cup \\
&\qquad \mathsf{Constraints}\Big(\overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma^{d'}}; \overrightarrow{\mathbb{C}_2}\Big) \\
&\mathsf{Constraints}\Big(\Gamma, \Theta, x : (\sigma^b)^e \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} x(V : \sigma^b)_{\underset{\mathrm{certify}}{?}}\mathrm{P} \oplus \mathrm{Q} : \mathsf{Proc}^{b\cdot c+\overline{b}\cdot d}; \\
&\qquad\qquad\qquad\qquad \mathbb{C}_{1V}\ \langle b\rangle\ \mathbb{C}_{2V}, x : (b \cdot c + \overline{b} \cdot d)\Big) = \\
&\qquad \{c \leq_{\mathsf{B}} \mathbb{C}_{1\mathrm{T}}(V), d \leq_{\mathsf{B}} \mathbb{C}_{2\mathrm{T}}(V)\} \cup \\
&\qquad \mathsf{Constraints}\big(\Gamma_V, V : \sigma^{\mathrm{T}}, x : (\sigma^b)^e \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^c; \mathbb{C}_1\big) \cup \\
&\qquad \mathsf{Constraints}\big(\Theta_V, V : \sigma^{\mathrm{U}}, x : (\sigma^b)^e \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{Q} : \mathsf{Proc}^d; \mathbb{C}_2\big)
\end{aligned}
$$

Figure 5.11: Constraints Generated by a Deduction in $\mathcal{T}_{\mathsf{HO}\pi\leq}$ (continued)

Theorem 5.5.3 (Soundness of $\{\![\cdot]\!\}$) *For every valid set of sub-type constraints $\overrightarrow{\sigma_1^b} \leq \overrightarrow{\sigma_2^c}$, if $\mathbb{S} = \{\![\overrightarrow{\sigma_1^b} \leq \overrightarrow{\sigma_2^c}]\!\}$ is defined, then*

- *For each $\mathsf{Proc}^b \leq \mathsf{Proc}^c$ in $\overrightarrow{\sigma_1^b} \leq \overrightarrow{\sigma_2^c}$, $\mathbb{S}(b) = \mathbb{S}(c)$*
- *For each $(\overrightarrow{\sigma_1^b}) \to \mathsf{Proc}^c \leq (\overrightarrow{\sigma_2^d}) \to \mathsf{Proc}^e$ in $\overrightarrow{\sigma_1^b} \leq \overrightarrow{\sigma_2^c}$, $\mathbb{S}(c) = \mathbb{S}(e)$*

Proof. By soundness of BUNIFY and inspection of the algorithm. □

Theorem 5.5.4 (Soundness of $\mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}$) *For any finite sequence of valid terms $\overrightarrow{K}$ in System $\mathcal{T}_{\mathsf{HO}\pi\leq}$, if*

$$\left\langle \overrightarrow{\Gamma}, \mathcal{C}, \overrightarrow{\sigma^b}, \overrightarrow{\mathbb{C}} \right\rangle = \mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}\left(\overrightarrow{K}\right)$$

is defined then

$$\overrightarrow{\Gamma} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}(\overrightarrow{K}) : \overrightarrow{\sigma^b}; \overrightarrow{\mathbb{C}}$$

and

$$\mathsf{Constraints}\left(\overrightarrow{\Gamma} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}(\overrightarrow{K}) : \overrightarrow{\sigma^b}; \overrightarrow{\mathbb{C}}\right) = \mathcal{C}$$

Proof. By cases on the structure of $\overrightarrow{K}$ and examination of the algorithm, appealing to the soundness of the algorithm $\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}$.

- $\overrightarrow{K} \equiv x$: From the algorithm $\mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(x) = \langle \{x : \alpha^i\}, \emptyset, \alpha^i, \emptyset, \mathsf{Id}\rangle$, and hence by the first variable type rule, $x : \alpha^i \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} x : \alpha^i; \emptyset$ and

  $$\mathsf{Constraints}\big(x : \alpha^i \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} x : \alpha^i; \emptyset\big) = \emptyset$$

  as required.

- $\overrightarrow{K} \equiv \mathrm{A}$: From the algorithm $\mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{A}) = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{A})$, and by Theorem 5.5.5 this is sound so soundness also holds for $\mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}$.

- $\overrightarrow{K} \equiv K'\overrightarrow{K''}$: By Theorem 5.5.5, $\left\langle \Gamma, \mathcal{C}, \sigma^b, \mathbb{C}, \mathbb{S}_1 \right\rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(K')$ implies $\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}_1(K') : \sigma^b; \mathbb{C}$ and $\mathsf{Constraints}\big(\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}_1(K') : \sigma^b; \mathbb{C}\big) = \mathcal{C}$. Then by the induction hypothesis,

  $$\left\langle \overrightarrow{\Gamma'}, \mathcal{C}', \overrightarrow{\sigma'^c}, \overrightarrow{\mathbb{C}'}, \mathbb{S}_2 \right\rangle = \mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}\left(\mathbb{S}_1(\overrightarrow{K'})\right)$$

  implies

  $$\overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}_2(\overrightarrow{K''}) : \overrightarrow{\sigma'^c}; \overrightarrow{\mathbb{C}'}$$

and

$$\mathsf{Constraints}\Big(\overrightarrow{\Gamma'}\mathbb{S}_2 \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} (\overrightarrow{K''}) : \overrightarrow{\sigma'^c}; \overrightarrow{\mathbb{C}'}\Big) = \mathcal{C}'$$

By soundness of Unify, $\mathbb{S}_3 = \mathbb{S}_1; \mathbb{S}_2; \mathsf{Unify}\Big(\Gamma, \bigcup \overrightarrow{\Gamma'}\Big)$ implies $\mathbb{S}_3(\Gamma) \asymp \mathbb{S}_3(\bigcup \overrightarrow{\Gamma'})$, and similarly by soundness of CUnify

$$\mathbb{S}_4 = \mathbb{S}_3; \mathsf{CUnify}\Big(\mathbb{S}_3(\mathbb{C}), \mathbb{S}_3(\bigcup \overrightarrow{\mathbb{C}'})\Big)$$

implies $\mathbb{S}_4(\mathbb{C}) \asymp \mathbb{S}_4(\bigcup \overrightarrow{\mathbb{C}'})$. Hence by definition,

$$\mathbb{S}_4(\Gamma\overrightarrow{\Gamma'}) \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}_4(K'\overrightarrow{K''}) : \mathbb{S}_4(\sigma^b\overrightarrow{\sigma'^c}); \mathbb{S}_4(\mathbb{C}\overrightarrow{\mathbb{C}'})$$

and

$$\mathsf{Constraints}\Big(\mathbb{S}_4(\Gamma\overrightarrow{\Gamma'}) \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}_4(K'\overrightarrow{K''}) : \mathbb{S}_4(\sigma^b\overrightarrow{\sigma'^c}); \mathbb{S}_4(\mathbb{C}\overrightarrow{\mathbb{C}'})\Big) = \mathbb{S}_4(\mathcal{C}, \mathcal{C}')$$

as required.

□

Theorem 5.5.5 (Soundness of $\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}$) *For every valid term* A *in System* $\mathcal{T}_{\mathsf{HO}\pi\leq}$,

$$\big\langle \Gamma, \mathcal{C}, \sigma^b, \mathbb{C}, \mathbb{S} \big\rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{A})$$

implies

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}(\mathrm{A}) : \sigma^b; \mathbb{C}$$

and

$$\mathsf{Constraints}\big(\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}(\mathrm{A}) : \sigma^b; \mathbb{C}\big) = \mathcal{C}$$

Proof. By induction on the structure of A and examination of the algorithm.

- $\mathrm{A} \equiv \mathbf{0}^b$: From the algorithm, $\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}\big(\mathbf{0}^b\big) = \big\langle \emptyset, \emptyset, \mathsf{Proc}^b, \emptyset, \mathsf{Id} \big\rangle$. By the zero type rule,

  $$\emptyset \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \mathbf{0}^b : \mathsf{Proc}^b; \emptyset$$

  and

  $$\mathsf{Constraints}\Big(\emptyset \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \mathbf{0}^b : \mathsf{Proc}^b; \emptyset\Big) = \emptyset$$

  by the definition of $\mathsf{Constraints}$ as required.

- A $\equiv X$: From the algorithm, $\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(X) = \langle\{X : \alpha^i\}, \emptyset, \alpha^i, \{X : i\}, \mathsf{Id}\rangle$. By the second variable introduction rule,

$$X : \alpha^i \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} X : \alpha^i; X : i$$

and

$$\mathsf{Constraints}\big(X : \alpha^i \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} X : \alpha^i; X : i\big) = \emptyset$$

by the definition of $\mathsf{Constraints}$ as required.

- A $\equiv$!P: From the algorithm, $\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(!\mathrm{P}) = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{P})$, and by the induction hypothesis $\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{P}) = \big\langle\Gamma, \mathcal{C}, \mathsf{Proc}^b, \mathbb{C}, \mathbb{S}\big\rangle$ implies

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}(\mathrm{P}) : \mathsf{Proc}^b; \mathbb{C}$$

and

$$\mathsf{Constraints}\big(\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}(\mathrm{P}) : \mathsf{Proc}^b; \mathbb{C}\big) = \mathcal{C}$$

Hence by the replication rule and the definition of $\mathsf{Constraints}$, soundness holds for !P as required.

- A $\equiv (\nu x : \sigma^c)$P: By the induction hypothesis,

$$\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{P}) = \big\langle\Gamma, \mathcal{C}, \mathsf{Proc}^b, \mathbb{C}, \mathbb{S}_1\big\rangle$$

implies

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}_1(\mathrm{P}) : \mathsf{Proc}^b; \mathbb{C}$$

and

$$\mathsf{Constraints}\big(\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}_1(\mathrm{P}) : \mathsf{Proc}^b; \mathbb{C}\big) = \mathcal{C}$$

By soundness of Unify, $\mathbb{S}_2 = \mathbb{S}_1; \mathsf{Unify}(\Gamma, \mathbb{S}_1(\{x : \sigma^c\}))$ implies $\mathbb{S}_2(\Gamma) \asymp \mathbb{S}_2(\{x : \sigma^c\})$, hence by the restriction rule and the definitions of the algorithm and $\mathsf{Constraints}$,

$$\mathbb{S}_2(\Gamma_x \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} (\nu x : \sigma^c)\mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_x)$$

and

$$\mathsf{Constraints}\big(\mathbb{S}_2(\Gamma_x \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} (\nu x : \sigma^c)\mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_x)\big) = \mathbb{S}_2(\mathcal{C})$$

as required.

- $\mathrm{A} \equiv \mathrm{P} + \mathrm{Q}$: By the induction hypothesis,

$$\left\langle \Gamma, \mathcal{C}_1, \mathsf{Proc}^b, \mathbb{C}_1, \mathbb{S}_1 \right\rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{P})$$

implies

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \mathbb{S}_1(\mathrm{P}) : \mathsf{Proc}^b; \mathbb{C}_1$$

and

$$\mathsf{Constraints}\Big(\Gamma \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \mathbb{S}_1(\mathrm{P}) : \mathsf{Proc}^b; \mathbb{C}_1\Big) = \mathcal{C}_1$$

Similarly,

$$\langle \Theta, \mathcal{C}_2, \mathsf{Proc}^c, \mathbb{C}_2, \mathbb{S}_2 \rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathbb{S}_1(\mathrm{Q}))$$

implies

$$\Theta \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \mathbb{S}_2(\mathrm{Q}) : \mathsf{Proc}^c; \mathbb{C}_2$$

and

$$\mathsf{Constraints}\Big(\Theta \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \mathbb{S}_2(\mathrm{Q}) : \mathsf{Proc}^c; \mathbb{C}_2\Big) = \mathcal{C}_2$$

Then, by soundness of Unify, $\mathbb{S}_3 = \mathbb{S}_1; \mathbb{S}_2; \mathsf{Unify}(\mathbb{S}_2(\Gamma), \Theta)$ implies $\mathbb{S}_3(\Gamma) \asymp \mathbb{S}_3(\Theta)$, and similarly by soundness of CUnify, $\mathbb{S}_4 = \mathbb{S}_3; \mathsf{CUnify}(\mathbb{S}_3(\mathbb{C}_1), \mathbb{S}_3(\mathbb{C}_2))$ implies $\mathbb{S}_4(\mathbb{C}_1) \asymp \mathbb{S}_4(\mathbb{C}_2)$. Hence by the summation type rule and definition of $\mathsf{Constraints}$,

$$\mathbb{S}_4(\Gamma, \Theta) \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \mathbb{S}_4(\mathrm{P} + \mathrm{Q}) : \mathsf{Proc}^{\mathbb{S}_4(bc)}; \mathbb{S}_4(\mathbb{C}_1, \mathbb{C}_2)$$

and

$$\mathsf{Constraints}\Big(\mathbb{S}_4(\Gamma, \Theta) \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \mathbb{S}_4(\mathrm{P} + \mathrm{Q}) : \mathsf{Proc}^{\mathbb{S}_4(bc)}; \mathbb{S}_4(\mathbb{C}_1, \mathbb{C}_2)\Big) = \mathbb{S}_4(\mathcal{C}_1, \mathcal{C}_2)$$

as required.

- $\mathrm{A} \equiv \mathrm{Q}|\mathrm{R}$: Similarly.

- $\mathrm{A} \equiv (\overrightarrow{V} : \overrightarrow{\sigma^c})\mathrm{P}$: By the induction hypothesis,

$$\left\langle \Gamma, \mathcal{C}, \mathsf{Proc}^b, \mathbb{C}, \mathbb{S}_1 \right\rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{P})$$

implies

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}_1(\mathrm{P}) : \mathsf{Proc}^b; \mathbb{C}$$

and

$$\mathsf{Constraints}\big(\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}_1(\mathrm{P}) : \mathsf{Proc}^b; \mathbb{C}\big) = \mathcal{C}$$

Then, by soundness of Unify, $\mathbb{S}_2 = \mathbb{S}_1; \mathsf{Unify}\Big(\Gamma, \mathbb{S}_1(\{\overrightarrow{V} : \overrightarrow{\sigma^c}\})\Big)$ implies $\mathbb{S}_2(\Gamma) \asymp \mathbb{S}_2(\{\overrightarrow{V} : \overrightarrow{\sigma^c}\})$, and hence by the abstraction rule and the definition of $\mathsf{Constraints}$,

$$\mathbb{S}_2(\Gamma_{\overrightarrow{V}} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} (\overrightarrow{V} : \overrightarrow{\sigma^c})\mathrm{P} : (\overrightarrow{\sigma^c}) \to \mathsf{Proc}^b; \mathbb{C}_{\overrightarrow{V}})$$

and

$$\mathsf{Constraints}\Big(\mathbb{S}_2(\Gamma_{\overrightarrow{V}} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} (\overrightarrow{V} : \overrightarrow{\sigma^c})\mathrm{P} : (\overrightarrow{\sigma^c}) \to \mathsf{Proc}^b; \mathbb{C}_{\overrightarrow{V}})\Big) = \mathbb{S}_2(\mathcal{C}, \forall V \in \overrightarrow{V}.b \leq_{\mathsf{B}} \mathbb{C}_{\mathrm{T}}(V))$$

as required.

- $\mathrm{A}' \equiv \mathrm{A}\langle\overrightarrow{K}\rangle$: By the induction hypothesis:

$$\langle \Gamma, \mathcal{C}_1, \sigma^c, \mathbb{C}_1, \mathbb{S}_1\rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{A})$$

implies

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}_1(\mathrm{A}) : \sigma^c; \mathbb{C}_1$$

and

$$\mathsf{Constraints}\big(\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}_1(\mathrm{A}) : \sigma^c; \mathbb{C}_1\big) = \mathcal{C}_1$$

Similarly, by soundness of $\mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}$

$$\Big\langle \overrightarrow{\Theta}, \mathcal{C}_2, \overrightarrow{\sigma'^d}, \overrightarrow{\mathbb{C}_2}, \mathbb{S}_2 \Big\rangle = \mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}\Big(\mathbb{S}_1(\overrightarrow{K})\Big)$$

implies

$$\overrightarrow{\Theta} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}_1; \mathbb{S}_2(\overrightarrow{K}) : \overrightarrow{\sigma'^d}; \overrightarrow{\mathbb{C}_2}$$

and

$$\mathsf{Constraints}\Big(\overrightarrow{\Theta} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}_1; \mathbb{S}_2(\overrightarrow{K}) : \overrightarrow{\sigma'^d}; \overrightarrow{\mathbb{C}_2}\Big) = \mathcal{C}_2$$

Then $\mathbb{S}_3 = \mathbb{S}_1; \mathbb{S}_2; \mathsf{Unify}\Big(\Gamma, \mathbb{S}_1; \mathbb{S}_2(\bigcup \overrightarrow{\Theta})\Big)$ implies $\mathbb{S}_3(\Gamma) \asymp \mathbb{S}_3(\bigcup \overrightarrow{\Theta})$ by soundness of Unify, and likewise $\mathbb{S}_4 = \mathbb{S}_3; \mathcal{U}\Big(\mathbb{S}_3(\sigma^c), (\overrightarrow{\alpha^i}) \to \mathsf{Proc}^{\mathbb{S}_3(b)}\Big)$ implies $\mathbb{S}_4(\sigma^c) = \mathbb{S}_4((\overrightarrow{\alpha^i}) \to \mathsf{Proc}^b)$ by the soundness of $\mathcal{U}$. Similarly, $\mathbb{S}_5 = \mathbb{S}_4; \mathsf{CUnify}\Big(\mathbb{S}_4(\mathbb{C}_1), \mathbb{S}_4(\bigcup \overrightarrow{\mathbb{C}_2}, \mathsf{chans}\Big(\overrightarrow{K}\Big) : c)\Big)$ implies

$$\mathbb{S}_5(\mathbb{C}_1) \asymp \mathbb{S}_5(\bigcup \overrightarrow{\mathbb{C}_2}, \mathsf{chans}\Big(\overrightarrow{K}\Big) : c)$$

by soundness of CUnify. Then by soundness of $\mathcal{U}$ again

$$\mathbb{S}_6 = \mathbb{S}_5; \mathcal{U}\Big(\mathbb{S}_5(\overrightarrow{\alpha^d}), \mathbb{S}_5(\overrightarrow{\sigma'^d})\Big)$$

is also sound, and similarly for $\mathbb{S}_7 = \mathbb{S}_6; \{\![\mathbb{S}_6(\overrightarrow{\sigma'^d} \leq \overrightarrow{\alpha^i})]\!\}$ by soundness of $\{\![\cdot]\!\}$ (noting that the base types are in the correct form by the previous statement). Then by the application type rule,

$$\mathbb{S}_7(\Gamma, \overrightarrow{\Theta} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{A}\langle\overrightarrow{K}\rangle : \mathsf{Proc}^b; \mathbb{C}_1, \overrightarrow{\mathbb{C}_2})$$

and

$$\begin{aligned}\mathsf{Constraints}\Big(\mathbb{S}_7(\Gamma, \overrightarrow{\Theta} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{A}\langle\overrightarrow{K}\rangle : \mathsf{Proc}^b; \mathbb{C}_1, \overrightarrow{\mathbb{C}_2})\Big) = \\ \mathbb{S}_7(\mathcal{C}_1, \mathcal{C}_2, [\![\overrightarrow{\sigma'^d} \leq \overrightarrow{\alpha^i}]\!], \\ \forall x \in \mathbb{C}_\epsilon.c \leq_{\mathsf{B}} \overrightarrow{\mathbb{C}_{2\mathrm{T}}}(x), \\ \forall \mathbb{C}_{2j}, V \in \mathrm{dom}\mathbb{C}_{2j}.\mathbb{C}_{2j} \leq_{\mathsf{B}} d_j)\end{aligned}$$

as required.

- $\mathrm{A} \equiv x(\overrightarrow{V} : \overrightarrow{\sigma^c}).\mathrm{P}$: By the induction hypothesis, the step

$$\big\langle \Gamma, \mathcal{C}, \mathsf{Proc}^b, \mathbb{C}, \mathbb{S}_1 \big\rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathrm{HO}\pi\leq}}(\mathrm{P})$$

implies both

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}_1(\mathrm{P}) : \mathsf{Proc}^b; \mathbb{C}$$

and

$$\mathsf{Constraints}\big(\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}_1(\mathrm{P}) : \mathsf{Proc}^b; \mathbb{C}\big) = \mathcal{C}$$

By the soundness of Unify, $\mathbb{S}_2 = \mathbb{S}_1; \mathsf{Unify}\Big(\Gamma, \mathbb{S}_1(\{x : (\overrightarrow{\sigma^j})^i, \overrightarrow{V} : \overrightarrow{\sigma^c}\})\Big)$ implies $\mathbb{S}_2(\Gamma) \asymp \mathbb{S}_2(\{x : (\overrightarrow{\sigma^j})^i, \overrightarrow{V} : \overrightarrow{\sigma^c}\})$. Then by soundness of $\mathbb{M}$, $\mathbb{S}_3 = \mathbb{S}_2; \mathbb{M}\Big(\mathbb{S}_2(i), \mathbb{S}_2(\overrightarrow{\sigma^j})\Big)$ is sound (as required by the well-formed type rule). By the soundness of CUnify, $\mathbb{S}_4 = \mathbb{S}_3; \mathsf{CUnify}(\mathbb{S}_3(\mathbb{C}), \mathbb{S}_3(\{x : b\}))$ implies $\mathbb{S}_4(\mathbb{C}) \asymp \mathbb{S}_4(\{x : b\})$ as required. Similarly, by soundness of $\{\![\cdot]\!\}$, $\mathbb{S}_5 = \mathbb{S}_4; \{\![\mathbb{S}_4(\overrightarrow{\sigma^j} \leq \overrightarrow{\sigma^c})]\!\}$ is sound, noting that the base types are in the correct form due to the explicit typing and the requirement that the term is valid. Then by the input type rule and the definition of $\mathsf{Constraints}$,

$$\mathbb{S}_5(\Gamma_{\overrightarrow{V}}, x : (\overrightarrow{\sigma^j})^i \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} x(\overrightarrow{V} : \overrightarrow{\sigma^c}).\mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_{\overrightarrow{V}}, x : b)$$

and

$$\mathsf{Constraints}\Big(\mathbb{S}_5(\Gamma_{\overrightarrow{V}}, x : (\overrightarrow{\sigma^j})^i \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} x(\overrightarrow{V} : \overrightarrow{\sigma^c}).\mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_{\overrightarrow{V}}, x : b)\Big) = \mathcal{C}, [\![\overrightarrow{\sigma^j} \leq \overrightarrow{\sigma^c}]\!], \forall V \in \overrightarrow{V}.b \leq_{\mathsf{B}} \mathbb{C}_{\mathrm{T}}(V)$$

as required.

- $\mathrm{A} \equiv \overline{x}[\overrightarrow{K}].\mathrm{P}$: By the induction hypothesis the first statement

$$\langle \Gamma, \mathcal{C}_1, \mathsf{Proc}^b, \mathbb{C}_1, \mathbb{S}_1 \rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{P})$$

implies both

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_1$$

and

$$\mathsf{Constraints}\big(\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_1\big) = \mathcal{C}_1$$

Similarly, by the soundness of $\mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}$,

$$\Big\langle \overrightarrow{\Theta}, \mathcal{C}_2, \overrightarrow{\sigma^c}, \overrightarrow{\mathbb{C}_2}, \mathbb{S}_2 \Big\rangle = \mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}\Big(\mathbb{S}_1(\overrightarrow{K})\Big)$$

implies

$$\overrightarrow{\Theta} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}_2(\overrightarrow{K}) : \overrightarrow{\sigma^c}; \overrightarrow{\mathbb{C}_2}$$

and

$$\mathsf{Constraints}\Big(\overrightarrow{\Theta} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}_2(\overrightarrow{K}) : \overrightarrow{\sigma^c}; \overrightarrow{\mathbb{C}_2}\Big) = \mathcal{C}_2$$

Then by the soundness of Unify, $\mathbb{S}_3 = \mathbb{S}_1; \mathbb{S}_2; \mathsf{Unify}\Big(\Gamma, \mathbb{S}_1(\bigcup \overrightarrow{\Theta})\Big)$ and $\mathbb{S}_4 = \mathbb{S}_3; \mathsf{Unify}\Big(\mathbb{S}_3(\Gamma, \overrightarrow{\Theta}), \{x : (\overrightarrow{\sigma^j})^i\}\Big)$ $\mathbb{S}_4(\Gamma, \overrightarrow{\Theta}, x : (\overrightarrow{\sigma^j})^i)$ is defined as required. Similarly, the soundness of $\mathbb{M}$ means that $\mathbb{S}_5 = \mathbb{S}_4; \mathbb{M}\Big(\mathbb{S}_4(b), \mathbb{S}_4(\overrightarrow{\sigma^j})\Big)$ and $\mathbb{S}_6 = \mathbb{S}_5; \mathbb{M}\Big(\mathbb{S}_3(i), \mathbb{S}_3(\overrightarrow{\sigma^j})\Big)$ ensures the types carried by x are multiplied by both the annotation of x and the process integrity of P as required by the type rule. In a similar fashion, the soundness of CUnify ensures that given $\mathbb{S}_7 = \mathbb{S}_6; \mathsf{CUnify}\Big(\mathbb{S}_6(\mathbb{C}_1), \mathbb{S}_6(\bigcup \overrightarrow{\mathbb{C}_2})\Big)$ and $\mathbb{S}_8 = \mathbb{S}_7; \mathsf{CUnify}\Big(\mathbb{S}_7(\mathbb{C}_1, \overrightarrow{\mathbb{C}_2}), \mathbb{S}_7(\mathsf{chans}\Big(\overrightarrow{K}\Big) : b, x : b)\Big)$ then $\mathbb{S}_8(\mathbb{C}_1, \overrightarrow{\mathbb{C}_2}, \mathsf{chans}\Big(\overrightarrow{K}\Big) : b, x : b)$ is defined. Likewise, by soundness of $\{[\![\cdot]\!]\}$ $\mathbb{S}_9 = \mathbb{S}_8; \{[\![\mathbb{S}_8(\overrightarrow{\sigma^c} \leq \overrightarrow{\sigma^j})]\!]\}$ is sound. Hence by the output type rule and the definition of $\mathsf{Constraints}$,

$$\mathbb{S}_9(\Gamma, \overrightarrow{\Theta}, x : (\overrightarrow{\sigma^j})^i \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \overline{x}[\overrightarrow{K}].\mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_1, \overrightarrow{\mathbb{C}_2}, \mathsf{chans}\Big(\overrightarrow{K}\Big) : b, x : b)$$

and

$$\mathsf{Constraints}\Big(\mathbb{S}_9(\Gamma, \overrightarrow{\Theta}, x : (\overrightarrow{\sigma^j})^i \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \overline{x}[\overrightarrow{K}].\mathrm{P} : \mathsf{Proc}^b; \mathbb{C}_1, \overrightarrow{\mathbb{C}_2}, \mathsf{chans}\Big(\overrightarrow{K}\Big) : b, x : b)\Big) = \mathbb{S}_9(\mathcal{C}_1, \mathcal{C}_2, [\![\overrightarrow{\sigma^c} \leq \overrightarrow{\sigma^j}]\!], \forall x \in \mathbb{C}_\epsilon.b \leq_{\mathsf{B}} \overrightarrow{\mathbb{C}_{2\mathrm{T}}}(x), \forall \mathbb{C}_{2k}, V \in \mathrm{dom}\mathbb{C}_{2j}.\mathbb{C}_{2k}(V) \leq_{\mathsf{B}} c_k)$$

as required.

- $\mathrm{A} \equiv x(V : \sigma^i)_{\mathrm{certify}}\mathbf{?}\mathrm{P} \oplus \mathrm{Q}$: By the induction hypothesis, the statement $\langle \Gamma, \mathcal{C}_1, \mathsf{Proc}^b, \mathbb{C}_1, \mathbb{S}_1 \rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{P})$ implies

$$\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}_1(\mathrm{P}) : \mathsf{Proc}^b; \mathbb{C}_1$$

and

$$\mathsf{Constraints}\big(\Gamma \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}_1(\mathrm{P}) : \mathsf{Proc}^b; \mathbb{C}_1\big) = \mathcal{C}_1$$

Similarly, $\langle \Theta, \mathcal{C}_2, \mathsf{Proc}^c, \mathbb{C}_2, \mathbb{S}_2 \rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathbb{S}_1(\mathrm{Q}))$ implies

$$\Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}_2(\mathrm{Q}) : \mathsf{Proc}^c; \mathbb{C}_2$$

and

$$\mathsf{Constraints}\big(\Theta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbb{S}_2(\mathrm{Q}) : \mathsf{Proc}^c; \mathbb{C}_2\big) = \mathcal{C}_2$$

Then by soundness of Unify,

$$\mathbb{S}_3 = \mathbb{S}_1; \mathbb{S}_2; \mathsf{Unify}\big(\Gamma, \mathbb{S}_1; \mathbb{S}_2(\{x : (\sigma^i)^j, V : \sigma^{\mathrm{T}}\})\big)$$

implies $\mathbb{S}_3(\Gamma, x : (\sigma^i)^j, V : \sigma^{\mathrm{T}})$ is defined, and likewise

$$\mathbb{S}_4 = \mathbb{S}_3; \mathsf{Unify}\big(\mathbb{S}_3(\Theta), \mathbb{S}_3(\{x : (\sigma^i)^j, V : \sigma^{\mathrm{U}}\})\big)$$

implies $\mathbb{S}_4(\Theta, x : (\sigma^i)^j, V : \sigma^{\mathrm{U}})$ is defined and

$$\mathbb{S}_5 = \mathbb{S}_4; \mathsf{Unify}(\mathbb{S}_4(\Gamma_V), \mathbb{S}_4(\Theta_V))$$

implies $\mathbb{S}_5(\Gamma_V, \Theta_V)$ is defined (noting that the bound variable V is removed during the unification as it has a different type in each). By the soundness of CUnify, $\mathbb{S}_6 = \mathbb{S}_5; \mathbb{M}(\mathbb{S}_5(j), \mathbb{S}_5(\sigma^i))$ means $\mathbb{S}_6(i) = \mathbb{S}_6(i \cdot j)$ as required by the well-formed type rule. Similarly,

$$\mathbb{S}_7 = \mathbb{S}_6; \mathsf{CUnify}\big(\mathbb{S}_6(\mathbb{C}_{1V} \langle i \rangle \mathbb{C}_{2V}), \mathbb{S}_6(\{x : i \cdot b + \bar{i} \cdot c\})\big)$$

is sound. Hence by the certify type rule and the definition of Constraints,

$$\mathbb{S}_7(\Gamma_V, \Theta_V, x : (\sigma^i)^j \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} x(V : \sigma^i)_{\underset{\text{certify}}{?}} \mathrm{P} \oplus \mathrm{Q} : \mathsf{Proc}^{ib+\bar{i}c};\ \mathbb{C}_{1V}\ \langle i \rangle\ \mathbb{C}_{2V}, x : i \cdot b + \bar{i} \cdot c)$$

and

$$\mathsf{Constraints}\Big(\mathbb{S}_7(\Gamma_V, \Theta_V, x : (\sigma^i)^j \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} x(V : \sigma^i)_{\underset{\text{certify}}{?}} \mathrm{P} \oplus \mathrm{Q} : \mathsf{Proc}^{ib+\bar{i}c};\ \mathbb{C}_{1V}\ \langle i \rangle\ \mathbb{C}_{2V}, x : i \cdot b + \bar{i} \cdot c)\Big) = \mathbb{S}_7(\mathcal{C}_1, \mathcal{C}_2, b \leq_{\mathsf{B}} \mathbb{C}_{1\mathrm{T}}(V), c \leq_{\mathsf{B}} \mathbb{C}_{2\mathrm{T}}(V))$$

as required.

□

Completeness

Definition 5.5.6 *For notational convenience, define the relation on tuples*

$$\langle \Gamma, \mathcal{C}, \sigma^c, \mathbb{C} \rangle \leq \langle \Gamma', \mathcal{C}', \sigma'^b, \mathbb{C}' \rangle$$

if and only if there is some substitution $\mathbb{S}$ *such that the following all hold:*

1. $\forall V \in \mathrm{dom}\Gamma.\mathbb{S}(\Gamma(V)) = \Gamma'(V)$ *(that is,* $\Gamma \leq \Gamma'$*)*
2. $\mathbb{S}(\mathcal{C}) \leq \mathcal{C}'$
3. $\mathbb{S}(\sigma^b) = \sigma'^c$
4. $\forall V \in \mathrm{dom}\mathbb{C}.\mathbb{S}(\mathbb{C}(V)) = \mathbb{C}'(V)$ *(that is,* $\mathbb{C} \leq \mathbb{C}'$*)*

Theorem 5.5.7 (Completeness of $\{[\cdot]\}$) *For every valid set of sub-type constraints* $\overrightarrow{\sigma_1^b} \leq \overrightarrow{\sigma_2^c}$*, if there is some substitution* $\mathbb{S}'$ *such that*

- *For each* $\mathsf{Proc}^b \leq \mathsf{Proc}^c$ *in* $\overrightarrow{\sigma_1^b} \leq \overrightarrow{\sigma_2^c}$, $\mathbb{S}'(b) = \mathbb{S}'(c)$
- *For each* $(\overrightarrow{\sigma_1^b}) \to \mathsf{Proc}^c \leq (\overrightarrow{\sigma_2^d}) \to \mathsf{Proc}^e$ *in* $\overrightarrow{\sigma_1^b} \leq \overrightarrow{\sigma_2^c}$, $\mathbb{S}'(c) = \mathbb{S}'(e)$

Then $\mathbb{S} = \{[\overrightarrow{\sigma_1^b} \leq \overrightarrow{\sigma_2^c}]\}$ *is defined, and* $\mathbb{S} \leq \mathbb{S}'$*.*

Proof. By completeness of BUNIFY. □

Theorem 5.5.8 (Completeness of $\mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}$)
$\forall \overrightarrow{\Gamma'}, \overrightarrow{K}, \overrightarrow{\sigma'^c}, \overrightarrow{\mathbb{C}'}, \mathcal{C}'$. *If there is a valid deduction in System* $\mathcal{T}_{\mathsf{HO}\pi\leq}$ *for*

$$\overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma'^c}; \overrightarrow{\mathbb{C}'}$$

where

$$\mathsf{Constraints}\Big(\overrightarrow{\Gamma'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma'^c}; \overrightarrow{\mathbb{C}'}\Big) = \mathcal{C}'$$

then

$$\Big\langle \overrightarrow{\Gamma}, \mathcal{C}, \overrightarrow{\sigma^b}, \overrightarrow{\mathbb{C}}, \mathbb{S} \Big\rangle = \mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}\Big(\overrightarrow{K}\Big)$$

is defined, and

$$\Big\langle \overrightarrow{\Gamma}, \mathcal{C}, \overrightarrow{\sigma^b}, \overrightarrow{\mathbb{C}} \Big\rangle \leq \Big\langle \overrightarrow{\Gamma'}, \mathcal{C}', \overrightarrow{\sigma'^c}, \overrightarrow{\mathbb{C}'} \Big\rangle$$

Proof. By induction on the length and contents of the derivation $\overrightarrow{\Gamma}' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma'^c}; \overrightarrow{\mathbb{C}'}$ and by examination of the algorithm, appealing to Theorem 5.5.9.

- $\Gamma' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} x : \sigma'^c; \emptyset$: By definition of the algorithm, $\mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(x) = \langle x : \alpha^i, \emptyset, \alpha^i, \emptyset\rangle$ which is defined, and $\langle x : \alpha^i, \emptyset, \alpha^i, \emptyset\rangle \leq \langle \Gamma', \emptyset, \sigma'^c, \emptyset\rangle$ for all values of Γ' and σ'^c (noting that $\mathsf{Constraints}\big(\Gamma' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} x : \sigma'^c; \emptyset\big) = \emptyset$).

- $\Gamma' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{A} : \sigma'^c; \mathbb{C}'$: By definition of the algorithm, $\mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{A}) = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{A})$, and by Theorem 5.5.9 the result holds for $\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{A})$.

- $\Gamma'_1\overrightarrow{\Gamma'_2} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} K\overrightarrow{K'} : \sigma'^{c_1}_1\overrightarrow{\sigma'^{c_2}_2}; \mathbb{C}'_1\overrightarrow{\mathbb{C}'_2}$: By Theorem 5.5.9,

  $$\mathsf{Constraints}\big(\Gamma'_1 \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} K : \sigma'^{c_1}_1; \mathbb{C}'_1\big) = \mathcal{C}'_1$$

  implies that

  $$\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(K) = \big\langle \Gamma_1, \mathcal{C}_1, \sigma^{b_1}_1, \mathbb{C}_1, \mathbb{S}_1\big\rangle$$

  is defined, and

  $$\big\langle \Gamma_1, \mathcal{C}_1, \sigma^{b_1}_1, \mathbb{C}_1\big\rangle \leq \big\langle \Gamma'_1, \mathcal{C}'_1, \sigma'^{c_1}_1, \mathbb{C}'_1\big\rangle$$

  Also, by the induction hypothesis,

  $$\mathsf{Constraints}\Big(\overrightarrow{\Gamma'_2} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K'} : \overrightarrow{\sigma'^{c_2}_2}; \overrightarrow{\mathbb{C}'_2}\Big) = \mathcal{C}'_2$$

  implies

  $$\mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}\Big(\overrightarrow{K'}\Big) = \Big\langle \overrightarrow{\Gamma_2}, \mathcal{C}_2, \overrightarrow{\sigma^{b_2}_2}, \overrightarrow{\mathbb{C}_2}, \mathbb{S}_2\Big\rangle$$

is defined, and

$$\left\langle \overrightarrow{\Gamma_2}, \mathcal{C}_2, \overrightarrow{\sigma_2^{b_2}}, \overrightarrow{\mathbb{C}_2} \right\rangle \leq \left\langle \overrightarrow{\Gamma'_2}, \mathcal{C}'_2, \overrightarrow{\sigma_2^{\prime c_2}}, \overrightarrow{\mathbb{C}'_2} \right\rangle$$

Then by completeness of Unify, $\mathbb{S}_3 = \mathbb{S}_1; \mathbb{S}_2; \mathsf{Unify}\Big(\Gamma, \bigcup \overrightarrow{\Gamma'}\Big)$ is defined and most general, as is $\mathbb{S}_4 = \mathbb{S}_3; \mathsf{CUnify}\Big(\mathbb{S}_3(\mathbb{C}), \mathbb{S}_3(\bigcup \overrightarrow{\mathbb{C}'})\Big)$ by soundness of CUnify. Hence as required,

$$\left\langle \mathbb{S}_4(\Gamma_1 \overrightarrow{\Gamma_2}), \mathbb{S}_4(\mathcal{C}_1, \mathcal{C}_2), \mathbb{S}_4(\sigma_1^{b_1} \overrightarrow{\sigma_2^{b_2}}), \mathbb{S}_4(\mathbb{C}_1 \overrightarrow{\mathbb{C}_2}) \right\rangle \leq \left\langle \Gamma'_1 \overrightarrow{\Gamma'_2}, (\mathcal{C}'_1, \mathcal{C}'_2), \sigma_1^{\prime c_1} \overrightarrow{\sigma_2^{\prime c_2}}, \mathbb{C}'_1 \overrightarrow{\mathbb{C}'_2} \right\rangle$$

□

Theorem 5.5.9 (Completeness of $\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}$)
$\forall \Gamma', \mathrm{P}, \sigma'^c, \mathbb{C}', \mathcal{C}'$. *If there is a valid deduction in System $\mathcal{T}_{\mathsf{HO}\pi\leq}$ for*

$$\Gamma' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{A} : \sigma'^c; \mathbb{C}'$$

with

$$\mathsf{Constraints}\big(\Gamma' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{A} : \sigma'^c; \mathbb{C}'\big) = \mathcal{C}'$$

then

$$\big\langle \Gamma, \mathcal{C}, \sigma^b, \mathbb{C}, \mathbb{S} \big\rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{A})$$

is defined, and

$$\big\langle \Gamma, \mathcal{C}, \sigma^b, \mathbb{C} \big\rangle \leq \big\langle \Gamma', \mathcal{C}', \sigma'^c, \mathbb{C}' \big\rangle$$

Proof. By cases on the structure of A then by induction on the derivation of $\Gamma' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{A} : \sigma^b; \mathbb{C}'$ and examination of the algorithm.

- $\mathrm{A} \equiv \mathbf{0}^b$: The deduction $\Gamma' \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \mathbf{0}^b : \mathsf{Proc}^b; \emptyset$ must have ended in a use of the zero type rule followed by zero or more uses of the weaken rule, and by examination of the algorithm,

  $$\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}\big(\mathbf{0}^b\big) = \big\langle \emptyset, \emptyset, \mathsf{Proc}^b, \emptyset, \mathsf{Id} \big\rangle$$

  Since $\mathsf{Constraints}\big(\Gamma' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathbf{0}^b : \mathsf{Proc}^b; \emptyset\big) = \emptyset$ for all Γ', by definition

  $$\big\langle \emptyset, \emptyset, \mathsf{Proc}^b, \emptyset \big\rangle \leq \big\langle \Gamma', \emptyset, \mathsf{Proc}^b, \emptyset \big\rangle$$

  as required.

- A $\equiv X$: The deduction $\Gamma' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} X : \sigma^c; X : c$ must have ended in a use of the second variable rule, possibly followed by zero or more applications of the weaken rule; by examination of the algorithm,

$$\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathrm{HO}\pi\leq}}(X) = \big\langle \{X : \alpha^i\}, \emptyset, \alpha^i, \{X : i\}, \mathsf{Id} \big\rangle$$

Since $\mathsf{Constraints}\big(\Gamma' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} X : \sigma^c; X : c\big) = \emptyset$, by definition

$$\big\langle \{X : \alpha^i\}, \emptyset, \alpha^i, \{X : i\} \big\rangle \leq \langle \Gamma', \emptyset, \sigma^c, \{X : c\} \rangle$$

as required (since by the second variable rule $\Gamma'(X) = \sigma^c$, and trivially $\exists \mathbb{S}.\mathbb{S}(\alpha^i) = \sigma^c$).

- A $\equiv$!P: The deduction $\Gamma', \Delta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} !\mathrm{P} : \mathsf{Proc}^c; \mathbb{C}'$ must have ended in zero or more uses of the weaken rule, preceded by a use of the replication rule with antecedent $\Gamma' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^c; \mathbb{C}'$ (where the domain of Δ is all the names and variables introduced via weakening after the final use of the replication rule). By the induction hypothesis,

$$\mathsf{Constraints}\big(\Gamma' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^c; \mathbb{C}'\big) = \mathcal{C}'$$

implies

$$\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathrm{HO}\pi\leq}}(\mathrm{P}) = \big\langle \Gamma, \mathcal{C}, \mathsf{Proc}^b, \mathbb{C}, \mathbb{S} \big\rangle$$

is defined, where $\big\langle \Gamma, \mathcal{C}, \mathsf{Proc}^b, \mathbb{C} \big\rangle \leq \langle \Gamma', \mathcal{C}', \mathsf{Proc}^c, \mathbb{C}' \rangle$. Hence by the definitions of the algorithm, the replication rule, and $\mathsf{Constraints}$, the result also holds for !P (noting that $\Gamma \leq \Gamma'$ implies $\Gamma \leq \Gamma', \Delta$ trivially).

- A $\equiv (\nu x : \sigma^b)$P: The deduction

$$\Gamma', x : \sigma^b, \Delta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} (\nu x : \sigma^b)\mathrm{P} : \mathsf{Proc}^{c'}; \mathbb{C}'_x$$

must have ended in zero or more uses of the weaken rule followed by an instance of the restriction rule with antecedent

$$\Gamma', x : \sigma^b \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{c'}; \mathbb{C}'$$

where the domain of Δ contains all names introduced via weakening, and assuming that $x \notin \mathrm{dom}\Gamma'$. By the induction hypothesis,

$$\mathsf{Constraints}\Big(\Gamma', x : \sigma^b \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{c'}; \mathbb{C}'\Big) = \mathcal{C}'$$

implies

$$\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathrm{HO}\pi\leq}}(\mathrm{P}) = \langle \Gamma, \mathcal{C}, \mathsf{Proc}^c, \mathbb{C}, \mathbb{S} \rangle$$

where

$$\langle \Gamma, \mathcal{C}, \mathsf{Proc}^c, \mathbb{C} \rangle \leq \langle \Gamma' \cup \{x : \sigma^b\}, \mathcal{C}', \mathsf{Proc}^c, \mathbb{C}' \rangle$$

Hence by the definitions of the algorithm, the replication rule, and $\mathsf{Constraints}$, the result also holds for

$$\langle \Gamma_x, \mathcal{C}, \mathsf{Proc}^c, \mathbb{C}_x \rangle \leq \left\langle \Gamma', \mathcal{C}', \mathsf{Proc}^{c'}, \mathbb{C}'_x \right\rangle$$

as required.

- $\mathrm{A} \equiv \mathrm{P} + \mathrm{Q}$: The deduction

$$\Gamma', \Theta', \Delta \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} \mathrm{P} + \mathrm{Q} : \mathsf{Proc}^{c_1 \cdot c_2}; \mathbb{C}'_1, \mathbb{C}'_2$$

must have ended in a zero or more uses of the weaken rule, preceded by an instance of the summation rule with antecedents

$$\begin{aligned} \Gamma' &\vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{c_1}; \mathbb{C}'_1 \\ \text{and} \quad \Theta' &\vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{Q} : \mathsf{Proc}^{c_2}; \mathbb{C}'_2 \end{aligned}$$

where the domain of Δ is all the names and variables introduced via weakening after the last application of the summation rule. By the induction hypothesis,

$$\mathsf{Constraints}\big(\Gamma' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{c_1}; \mathbb{C}'_1\big) = \mathcal{C}'_1$$

implies that

$$\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{P}) = \langle \Gamma, \mathcal{C}_1, \mathsf{Proc}^{b_1}, \mathbb{C}_1, \mathbb{S}_1 \rangle$$

is defined, where $\langle \Gamma, \mathcal{C}_1, \mathsf{Proc}^{b_1}, \mathbb{C}_1 \rangle \leq \langle \Gamma', \mathcal{C}'_1, \mathsf{Proc}^{c_1}, \mathbb{C}'_1 \rangle$, and

$$\mathsf{Constraints}\big(\Theta' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{Q} : \mathsf{Proc}^{c_2}; \mathbb{C}'_2\big) = \mathcal{C}'_2$$

implies

$$\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathbb{S}_2(\mathrm{Q})) = \langle \Theta, \mathcal{C}_2, \mathsf{Proc}^{b_2}, \mathbb{C}_2, \mathbb{S}_2 \rangle$$

where $\langle \Theta, \mathcal{C}_2, \mathsf{Proc}^{b_2}, \mathbb{C}_2 \rangle \leq \langle \Theta', \mathcal{C}'_2, \mathsf{Proc}^{c_2}, \mathbb{C}'_2 \rangle$.

By the completeness of Unify, since $\Gamma \leq \Gamma'$ and $\Theta \leq \Theta'$ (and $\Gamma' \asymp \Theta'$), then $\mathbb{S}_3 = \mathbb{S}_1; \mathbb{S}_2; \mathsf{Unify}(\mathbb{S}_2(\Gamma), \Theta)$ is defined and most general; similarly, $\mathbb{S}_4 = \mathbb{S}_3; \mathsf{CUnify}(\mathbb{S}_3(\mathbb{C}_1), \mathbb{S}_3(\mathbb{C}_2))$ is also defined and most general by the completeness of CUnify. Hence by the summation rule and the definition of the algorithm,

$$\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{P} + \mathrm{Q}) = \left\langle \mathbb{S}_4(\Gamma, \Theta), \mathbb{S}_4(\mathcal{C}_1, \mathcal{C}_2), \mathsf{Proc}^{\mathbb{S}_4(b_1 \cdot b_2)}, \mathbb{S}_4(\mathbb{C}_1, \mathbb{C}_2), \mathbb{S}_4 \right\rangle$$

is defined, and furthermore by definition of $\mathsf{Constraints}$

$$\mathsf{Constraints}\big(\Gamma', \Theta', \Delta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} + \mathrm{Q} : \mathsf{Proc}^{c_1 \cdot c_2}; \mathbb{C}'_1, \mathbb{C}'_2\big) = \mathcal{C}'_1, \mathcal{C}'_2$$

and

$$\Big\langle \mathbb{S}_4(\Gamma, \Theta), \mathbb{S}_4(\mathcal{C}_1, \mathcal{C}_2), \mathsf{Proc}^{\mathbb{S}_4(b_1 \cdot b_2)}, \mathbb{S}_4(\mathbb{C}_1, \mathbb{C}_2) \Big\rangle \\ \leq \langle (\Gamma', \Theta', \Delta), (\mathcal{C}'_1, \mathcal{C}'_2), \mathsf{Proc}^{c_1 \cdot c_2}, (\mathbb{C}'_1, \mathbb{C}'_2) \rangle$$

as required.

- $\mathrm{A} \equiv \mathrm{P}|\mathrm{Q}$: Similarly.
- $\mathrm{A} \equiv (\overrightarrow{V} : \overrightarrow{\sigma^b})\mathrm{P}$: The deduction

$$\Gamma', \Delta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} (\overrightarrow{V} : \overrightarrow{\sigma^b})\mathrm{P} : (\overrightarrow{\sigma^b}) \to \mathsf{Proc}^{c'}; \mathbb{C}'_{\overrightarrow{V}}$$

must have ended in zero or more uses of the weaken rule, preceded by an instance of the abstraction rule with antecedent

$$\Gamma', \overrightarrow{V} : \overrightarrow{\sigma^b} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{c'}; \mathbb{C}'$$

where the domain of Δ is all the names and variables introduced via weakening after the application of the abstraction rule, and assuming that $\overrightarrow{V}$ is not in the domain of Γ'. By the induction hypothesis,

$$\mathsf{Constraints}\Big(\Gamma', \overrightarrow{V} : \overrightarrow{\sigma^b} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{c'}; \mathbb{C}'\Big) = \mathcal{C}'$$

implies that

$$\langle \Gamma, \mathcal{C}, \mathsf{Proc}^c, \mathbb{C}, \mathbb{S}_1 \rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathrm{HO}\pi\leq}}(\mathrm{P})$$

is defined, and

$$\langle \Gamma, \mathcal{C}, \mathsf{Proc}^c, \mathbb{C} \rangle \leq \left\langle \Gamma'_{\overrightarrow{V}} \cup \{\overrightarrow{V} : \overrightarrow{\sigma^{b'}}\}, \mathcal{C}', \mathsf{Proc}^{c'}, \mathbb{C}' \right\rangle$$

Now, by completeness of Unify,

$$\mathbb{S}_2 = \mathbb{S}_1; \mathsf{Unify}\Big(\Gamma, \mathbb{S}_1(\{\overrightarrow{V} : \overrightarrow{\sigma^b}\})\Big)$$

is defined and most general. Hence by the abstraction rule and the definitions of the algorithm and $\mathsf{Constraints}$,

$$\Big\langle \mathbb{S}_2(\Gamma_{\overrightarrow{V}}), \mathbb{S}_2(\mathcal{C}, \forall V \in \overrightarrow{V}.c \leq_{\mathsf{B}} \mathbb{C}_{\mathrm{T}}(V)), \mathbb{S}((\overrightarrow{\sigma^b}) \to \mathsf{Proc}^c), \mathbb{S}_2(\mathbb{C}_{\overrightarrow{V}}) \Big\rangle \\ \leq \quad \Big\langle \Gamma' \cup \Delta, (\mathcal{C}', \forall V \in \overrightarrow{V}.c' \leq_{\mathsf{B}} \mathbb{C}'_{\mathrm{T}}(V)), (\overrightarrow{\sigma^b}) \to \mathsf{Proc}^{c'}, \mathbb{C}'_{\overrightarrow{V}} \Big\rangle$$

as required.

- $\mathrm{A}' \equiv \mathrm{A}\langle\overrightarrow{K}\rangle$: The deduction

$$\Gamma', \overrightarrow{\Theta'}, \Delta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{A}\langle\overrightarrow{K}\rangle : \mathsf{Proc}^{c'}; \mathbb{C}'_1, \overrightarrow{\mathbb{C}'_2}, \mathsf{chans}\left(\overrightarrow{K}\right) : c'$$

must have ended in zero or more uses of the weaken rule, preceded by an instance of the application rule with antecedents

$$\begin{aligned} \Gamma' &\vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{F} : (\overrightarrow{\sigma_1'^{b'}}) \to \mathsf{Proc}^{c'}; \mathbb{C}'_1 \\ \text{and} \quad \overrightarrow{\Theta'} &\vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma_2'^{d'}}; \overrightarrow{\mathbb{C}'_1} \end{aligned}$$

where $\overrightarrow{\sigma_1'^{d'}} \leq \overrightarrow{\sigma_1'^{b'}}$, and the domain of Δ is all the names and variables introduced by weakening after the use of the application rule. By the induction hypothesis,

$$\mathsf{Constraints}\left(\Gamma' \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{F} : (\overrightarrow{\sigma_1'^{b'}}) \to \mathsf{Proc}^{c'}; \mathbb{C}'_1\right) = \mathcal{C}'_1$$

implies

$$\langle\Gamma, \mathcal{C}_1, \sigma^c, \mathbb{C}_1, \mathbb{S}_1\rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathrm{A})$$

is defined, and

$$\langle\Gamma, \mathcal{C}_1, \sigma^c, \mathbb{C}_1\rangle \leq \left\langle\Gamma', \mathcal{C}'_1, (\overrightarrow{\sigma_1'^{b'}}) \to \mathsf{Proc}^{c'}, \mathbb{C}'_1\right\rangle$$

Similarly, by Theorem 5.5.8 (completeness of $\mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}$),

$$\mathsf{Constraints}\left(\overrightarrow{\Theta'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma_1'^{d'}}; \overrightarrow{\mathbb{C}'_2}\right) = \mathcal{C}'_2$$

implies

$$\left\langle\overrightarrow{\Theta}, \mathcal{C}_2, \overrightarrow{\sigma_2^d}, \overrightarrow{\mathbb{C}_2}, \mathbb{S}_2\right\rangle = \mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}\left(\mathbb{S}_1(\overrightarrow{K})\right)$$

is defined, and

$$\left\langle\overrightarrow{\Theta}, \mathcal{C}_2, \overrightarrow{\sigma_2^d}, \overrightarrow{\mathbb{C}_2}\right\rangle \leq \left\langle\overrightarrow{\Theta'}, \mathcal{C}'_2, \overrightarrow{\sigma_1'^{d'}}, \overrightarrow{\mathbb{C}'_2}\right\rangle$$

Now, by completeness of Unify, $\mathbb{S}_3 = \mathbb{S}_1; \mathbb{S}_2; \mathsf{Unify}\left(\Gamma, \mathbb{S}_1; \mathbb{S}_2(\bigcup \overrightarrow{\Theta})\right)$ is defined and most general. Similarly, by completeness of $\mathcal{U}$, $\mathbb{S}_4 = \mathbb{S}_3; \mathcal{U}\left(\mathbb{S}_3(\sigma^c), (\overrightarrow{\alpha^i}) \to \mathsf{Proc}^{\mathbb{S}_3(c)}\right)$ is defined and most general, and by

completeness of $\mathsf{CUnify}\ \mathbb{S}_5 = \mathbb{S}_4; \mathsf{CUnify}\Big(\mathbb{S}_4(\mathbb{C}_1), \mathbb{S}_4(\bigcup \overrightarrow{\mathbb{C}_2}, \mathsf{chans}\Big(\overrightarrow{K}\Big) : c)\Big)$ is defined and most general. By completeness of $\mathcal{U}$ again, the statement $\mathbb{S}_6 = \mathbb{S}_5; \mathcal{U}\Big(\mathbb{S}_5(\overrightarrow{\alpha^d}), \mathbb{S}_5(\overrightarrow{\sigma_2^d})\Big)$ is also defined and most general, noting that only the base types are unified, to preserve the constraints. Noting that the base types are therefore in the correct form, $\mathbb{S}_7 = \mathbb{S}_6; \{\!|\mathbb{S}_6(\overrightarrow{\sigma_2^d} \leq \overrightarrow{\alpha^i})|\!\}$ is defined and most-general by completeness of $\{\!|\cdot|\!\}$. Then by the application rule

$$\begin{aligned}
&\Big\langle \mathbb{S}_7(\Gamma, \overrightarrow{\Theta}), \mathbb{S}_7(\mathcal{C}_1, \mathcal{C}_2, [\![\overrightarrow{\sigma'^d} \leq \overrightarrow{\alpha^i}]\!], \forall x \in \mathbb{C}_\epsilon.c \leq_{\mathsf{B}} \overrightarrow{\mathbb{C}_{2\mathrm{T}}}(x), \\
&\qquad\qquad \forall \mathbb{C}_{2j}, V \in \mathrm{dom}\mathbb{C}_{2j}.\mathbb{C}_{2j} \leq_{\mathsf{B}} d_j), \\
&\qquad\qquad \mathbb{S}_7(\mathsf{Proc}^c), \mathbb{S}_7(\mathbb{C}_1, \overrightarrow{\mathbb{C}_2}, \mathsf{chans}\Big(\overrightarrow{K}\Big) : c)\Big\rangle \\
\leq\ &\Big\langle (\Gamma', \overrightarrow{\Theta'}), (\mathcal{C}_1', \mathcal{C}_2', \overrightarrow{b'} \leq_{\mathsf{B}} \overrightarrow{d'}, \forall x \in \mathbb{C}_\epsilon.\overrightarrow{\mathbb{C}_{2\mathrm{T}}'}(x) \geq_{\mathsf{B}} c', \\
&\qquad\qquad \forall \mathbb{C}_{2j}', V \in \mathrm{dom}\mathbb{C}_{2j}'.\mathbb{C}_{2j}'(C) \leq_{\mathsf{B}} d_j'), \\
&\qquad\qquad \mathsf{Proc}^{c'}, (\mathbb{C}_1, \overrightarrow{\mathbb{C}_2'}, \mathsf{chans}\Big(\overrightarrow{K}\Big) : c')\Big\rangle
\end{aligned}$$

as required.

- $\mathrm{A} \equiv x(\overrightarrow{V} : \overrightarrow{\sigma^b}).\mathrm{P}$: The deduction

$$\Gamma', x : (\overrightarrow{\sigma^d})^e, \Delta \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} x(\overrightarrow{V} : \overrightarrow{\sigma^b}).\mathrm{P} : \mathsf{Proc}^{c'}; \mathbb{C}'_{\overrightarrow{V}}, x : c'$$

must have ended in zero or more uses of the weaken rule, preceded by an instance of the input rule with antecedent

$$\Gamma', x : (\overrightarrow{\sigma^d})^e, \overrightarrow{V} : \overrightarrow{\sigma^b} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{c'}; \mathbb{C}'$$

where $\overrightarrow{\sigma^d} \leq \overrightarrow{\sigma^b}$ and the domain of Δ is all names and variables introduced via weakening after the application of the input rule, and assuming $\overrightarrow{V} \notin \mathrm{dom}\Gamma'$. By the induction hypothesis, given

$$\mathsf{Constraints}\Big(\Gamma', x : (\overrightarrow{\sigma^d})^e, \overrightarrow{V} : \overrightarrow{\sigma^b} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{c'}; \mathbb{C}'\Big) = \mathcal{C}'$$

then $\langle \Gamma, \mathcal{C}, \mathsf{Proc}^c, \mathbb{C}, \mathbb{S}_1 \rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathrm{HO}\pi\leq}}(\mathrm{P})$ is defined and

$$\langle \Gamma, \mathcal{C}, \mathsf{Proc}^c, \mathbb{C} \rangle \leq \Big\langle \Gamma' \cup \{x : (\overrightarrow{\sigma^d})^e, \overrightarrow{V} : \overrightarrow{\sigma^b}\}, \mathcal{C}', \mathsf{Proc}^{c'}, \mathbb{C}' \Big\rangle$$

Hence, by completeness of Unify, $\mathbb{M}$, and CUnify respectively the following statements are all defined and most-general:

- $\mathbb{S}_2 = \mathbb{S}_1; \mathsf{Unify}\Big(\Gamma, \mathbb{S}_1(\{x : (\overrightarrow{\sigma^j})^i, \overrightarrow{V} : \overrightarrow{\sigma^b}\})\Big)$
- $\mathbb{S}_3 = \mathbb{S}_2; \mathbb{M}\Big(\mathbb{S}_2(i), \mathbb{S}_2(\overrightarrow{\sigma^j})\Big)$
- $\mathbb{S}_4 = \mathbb{S}_3; \mathsf{CUnify}(\mathbb{S}_3(\mathbb{C}), \mathbb{S}_3(\{x : c\}))$

Therefore, since the base types are in the correct form, by completeness of $\{[\cdot]\}\ \mathbb{S}_5 = \mathbb{S}_4; \{[\mathbb{S}_4(\overrightarrow{\sigma^j} \leq \overrightarrow{\sigma^b})]\}$ is defined and most-general. Then by the input rule and the definition of $\mathsf{Constraints}$,

$$\Big\langle \mathbb{S}_5(\Gamma_{\overrightarrow{V}}, x : (\overrightarrow{\sigma^j})^i), \mathbb{S}_5(\mathcal{C}, [\![\overrightarrow{\sigma^j} \leq \overrightarrow{\sigma^b}]\!], \forall V \in \overrightarrow{V}.c \leq_{\mathsf{B}} \mathbb{C}_{\mathrm{T}}(V)), \\ \mathbb{S}_5(\mathsf{Proc}^c), \mathbb{S}_5(\mathbb{C}_{\overrightarrow{V}}, x : c)\Big\rangle \leq \\ \Big\langle (\Gamma', x : (\overrightarrow{\sigma^d})^e, \Delta), (\mathcal{C}', [\![\overrightarrow{\sigma^d} \leq \overrightarrow{\sigma^b}]\!], \forall V \in \overrightarrow{V}.c' \leq_{\mathsf{B}} \mathbb{C}'_{\mathrm{T}}(V)), \\ \mathsf{Proc}^{c'}, (\mathbb{C}'_{\overrightarrow{V}}, x : c')\Big\rangle$$

as required.

- $\mathrm{A} \equiv \overline{x}[\overrightarrow{K}].\mathrm{P}$: The deduction

$$\Gamma', x : (c' \cdot \overrightarrow{\sigma'^{d'}})^{b'}, \overrightarrow{\Theta'}, \Delta \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overline{x}[\overrightarrow{K}].\mathrm{P} : \mathsf{Proc}^{c'}; \\ \mathbb{C}'_1, x : c', \overrightarrow{\mathbb{C}'_2}, \mathsf{chans}\Big(\overrightarrow{K}\Big) : c'$$

must have ended in zero or more uses of the weaken rule, preceded by a use of the output rule with antecedents

$$\Gamma', x : (c' \cdot \overrightarrow{\sigma'^{d'}})^{b'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{c'}; \mathbb{C}'_1 \\ \text{and} \quad \overrightarrow{\Theta'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma'^{d''}}; \overrightarrow{\mathbb{C}'_2}$$

where $\overrightarrow{\sigma'^{d''}} \leq c' \cdot \overrightarrow{\sigma'^{d'}}$ and the domain of Δ is all the names and variables introduced via weakening after the last application of the output rule. By the induction hypothesis, given

$$\mathsf{Constraints}\Big(\Gamma', x : (c' \cdot \overrightarrow{\sigma'^{d'}})^{b'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{c'}; \mathbb{C}'_1\Big) = \mathcal{C}'_1$$

then

$$\langle \Gamma, \mathcal{C}_1, \mathsf{Proc}^c, \mathbb{C}_1, \mathbb{S}_1 \rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathrm{HO}\pi\leq}}(\mathrm{P})$$

is defined, and

$$\langle \Gamma, \mathcal{C}_1, \mathsf{Proc}^c, \mathbb{C}_1 \rangle \leq \Big\langle \Gamma', \mathcal{C}'_1, \mathsf{Proc}^{c'}, \mathbb{C}'_1 \Big\rangle$$

Similarly, by completeness of $\mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}$

$$\mathsf{Constraints}\Big(\overrightarrow{\Theta'} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \overrightarrow{K} : \overrightarrow{\sigma'^{d''}}; \overrightarrow{\mathbb{C}'_2}\Big) = \mathcal{C}'_2$$

implies

$$\Big\langle \overrightarrow{\Theta}, \mathcal{C}_2, \overrightarrow{\sigma^d}, \overrightarrow{\mathbb{C}_2}, \mathbb{S}_2 \Big\rangle = \mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}\Big(\mathbb{S}_1(\overrightarrow{K})\Big)$$

is defined, and

$$\Big\langle \overrightarrow{\Theta}, \mathcal{C}_2, \overrightarrow{\sigma^d}, \overrightarrow{\mathbb{C}_2} \Big\rangle \leq \Big\langle \overrightarrow{\Theta'}, \mathcal{C}'_2, \overrightarrow{\sigma'^{d''}}, \overrightarrow{\mathbb{C}'_2} \Big\rangle$$

Then by completeness of Unify, both

$$\mathbb{S}_3 = \mathbb{S}_1; \mathbb{S}_2; \mathsf{Unify}\Big(\Gamma, \mathbb{S}_1(\bigcup \overrightarrow{\Theta})\Big)$$

and

$$\mathbb{S}_4 = \mathbb{S}_3; \mathsf{Unify}\Big(\mathbb{S}_3(\Gamma, \overrightarrow{\Theta}), \{x : (\overrightarrow{\sigma^j})^i\}\Big)$$

are defined and most general. Likewise, by completeness of $\mathbb{M}$, both

$$\mathbb{S}_5 = \mathbb{S}_4; \mathbb{M}\Big(\mathbb{S}_4(c), \mathbb{S}_4(\overrightarrow{\sigma^j})\Big) \quad \text{and} \quad \mathbb{S}_6 = \mathbb{S}_5; \mathbb{M}\Big(\mathbb{S}_3(i), \mathbb{S}_3(\overrightarrow{\sigma^j})\Big)$$

are defined and most general (noting that the combination of the well-formed type rule with the output rule requires the object types to be multiplied by both the channel annotation and the process annotation). Lastly, by completeness of CUnify, both

$$\mathbb{S}_7 = \mathbb{S}_6; \mathsf{CUnify}\Big(\mathbb{S}_6(\mathbb{C}_1), \mathbb{S}_6(\bigcup \overrightarrow{\mathbb{C}_2})\Big)$$

and

$$\mathbb{S}_8 = \mathbb{S}_7; \mathsf{CUnify}\Big(\mathbb{S}_7(\mathbb{C}_1, \overrightarrow{\mathbb{C}_2}), \mathbb{S}_7(\mathsf{chans}\Big(\overrightarrow{K}\Big) : c, x : c)\Big)$$

are defined and most general. By completeness of $\{[\![\cdot]\!]\}$, $\mathbb{S}_9 = \mathbb{S}_8; \{[\![\mathbb{S}_8(\overrightarrow{\sigma^d} \leq \overrightarrow{\sigma^j})]\!]\}$ is also defined and most general. Then by the output rule and

the definitions of $\mathsf{Constraints}$ and the algorithm,

$$\begin{aligned}
&\Big\langle \mathbb{S}_9(\Gamma, \Theta, x : (\overrightarrow{\sigma^i})^j),\\
&\quad \mathbb{S}_9(\mathcal{C}_1, \mathcal{C}_2, [\![\overrightarrow{\sigma^d} \leq_{\mathsf{B}} \overrightarrow{\sigma^i}]\!], \forall x \in \mathbb{C}_\epsilon.c \leq_{\mathsf{B}} \overrightarrow{\mathbb{C}_{2\mathrm{T}}}(x)\\
&\qquad\qquad \forall \mathbb{C}_{2k}, V \in \mathrm{dom}\mathbb{C}_{2k}.\mathbb{C}_{2k}(V) \leq_{\mathsf{B}} d_k),\\
&\quad \mathbb{S}_9(\mathsf{Proc}^c), \mathbb{S}_9(\mathbb{C}_1, \bigcup \overrightarrow{\mathbb{C}_2}, x : c, \mathsf{chans}\Big(\overrightarrow{K}\Big) : c)\Big\rangle\\
\leq\\
&\Big\langle (\Gamma', \overrightarrow{\Theta'}, x : (c' \cdot \overrightarrow{\sigma'^{d'}})^{b'}, \Delta),\\
&\quad (\mathcal{C}'_1, \mathcal{C}'_2, [\![\overrightarrow{\sigma'^{d''}} \leq c' \cdot \overrightarrow{\sigma'^{d'}}]\!], \forall x \in \mathbb{C}_\epsilon.c' \leq_{\mathsf{B}} \mathbb{C}'_{2\mathrm{T}}(x),\\
&\qquad\qquad \forall \mathbb{C}'_{2k}, V \in \mathrm{dom}\mathbb{C}'_{2k}.\mathbb{C}'_{2k} \leq_{\mathsf{B}} d''_k)\\
&\qquad\qquad \mathsf{Proc}^{c'}, (\mathbb{C}'_1, x : c', \overrightarrow{\mathbb{C}'_2}, \mathsf{chans}\Big(\overrightarrow{K}\Big) : c')\Big\rangle
\end{aligned}$$

as required.

- $\mathrm{A} \equiv x(V : \sigma^i)_{\underset{\text{certify}}{?}}\mathrm{P} \oplus \mathrm{Q}$: The deduction

$$\begin{aligned}
\Gamma', \Theta', x : (\sigma^i)^d, \Delta \vdash^{\mathbb{C}_\epsilon}_{(\mathrm{HO}\leq)} x(V : \sigma^i)_{\underset{\text{certify}}{?}}\mathrm{P} \oplus \mathrm{Q} : \mathsf{Proc}^{i \cdot c_1 + \bar{i} \cdot c_2};\\
\mathbb{C}'_{1V} \langle i \rangle \mathbb{C}'_{2V}, x : (i \cdot c_1 + \bar{i} \cdot c_2)
\end{aligned}$$

must have ended in zero or more uses of the weaken rule, preceded by an instance of the certify rule with antecedents

$$\begin{aligned}
\Gamma', x : (\sigma^i)^d, V : \sigma^{\mathrm{T}} \quad &\vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \quad \mathrm{P} : \mathsf{Proc}^{c_1}; \mathbb{C}'_1\\
\text{and} \quad \Theta', x : (\sigma^i)^d, V : \sigma^{\mathrm{U}} \quad &\vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \quad \mathrm{Q} : \mathsf{Proc}^{c_2}; \mathbb{C}'_2
\end{aligned}$$

where the domain of Δ is all the names and variables introduced via weakening subsequent to the application of the certify rule (and assuming that V is not in the domain of either Γ' or Θ'). By the induction hypothesis,

$$\mathsf{Constraints}\big(\Gamma', x : (\sigma^i)^d, V : \sigma^{\mathrm{T}} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{P} : \mathsf{Proc}^{c_1}; \mathbb{C}'_1\big) = \mathcal{C}'_1$$

implies that

$$\big\langle \Gamma, \mathcal{C}_1, \mathsf{Proc}^{b_1}, \mathbb{C}_1, \mathbb{S}_1 \big\rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathrm{HO}\pi\leq}}(\mathrm{P})$$

is defined, and

$$\big\langle \Gamma, \mathcal{C}_1, \mathsf{Proc}^{b_1}, \mathbb{C}_1 \big\rangle \leq \big\langle \Gamma' \cup \{x : (\sigma^i)^d, V : \sigma^{\mathrm{T}}\}, \mathcal{C}'_1, \mathsf{Proc}^{c_1}, \mathbb{C}'_1 \big\rangle$$

Similarly,

$$\mathsf{Constraints}\big(\Theta', x : (\sigma^i)^d, V : \sigma^{\mathrm{U}} \vdash^{\mathbb{C}_\epsilon}_{\mathrm{HO}\leq} \mathrm{Q} : \mathsf{Proc}^{c_2}; \mathbb{C}'_2\big) = \mathcal{C}'_2$$

implies that

$$\left\langle \Theta, \mathcal{C}_2, \mathsf{Proc}^{b_2}, \mathbb{C}_2, \mathbb{S}_2 \right\rangle = \mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}(\mathbb{S}_1(\mathrm{Q}))$$

is defined, and

$$\left\langle \Theta, \mathcal{C}_2, \mathsf{Proc}^{b_2}, \mathbb{C}_2 \right\rangle \leq \left\langle \Theta' \cup \{x : (\sigma^i)^d, V : \sigma^{\mathrm{U}}\}, \mathcal{C}'_2, \mathsf{Proc}^{c_2}, \mathbb{C}'_2 \right\rangle$$

Now, by completeness of Unify, the following are all defined and most general:

- $\mathbb{S}_3 = \mathbb{S}_1; \mathbb{S}_2; \mathsf{Unify}\big(\Gamma, \mathbb{S}_1; \mathbb{S}_2(\{x : (\sigma^i)^j, V : \sigma^{\mathrm{T}}\})\big)$
- $\mathbb{S}_4 = \mathbb{S}_3; \mathsf{Unify}\big(\mathbb{S}_3(\Theta), \mathbb{S}_3(\{x : (\sigma^i)^j, V : \sigma^{\mathrm{U}}\})\big)$
- $\mathbb{S}_5 = \mathbb{S}_4; \mathsf{Unify}\big(\mathbb{S}_4(\Gamma_V), \mathbb{S}_4(\Theta_V)\big)$

(Noting that the unification is performed in the absence of V in the final step, due to the conflicting types). Similarly, by completeness of $\mathbb{M}$, $\mathbb{S}_6 = \mathbb{S}_5; \mathbb{M}(\mathbb{S}_5(j), \mathbb{S}_5(\sigma^i))$ is also defined and most general. Finally, by completeness of CUnify,

$$\mathbb{S}_7 = \mathbb{S}_6; \mathsf{CUnify}\big(\mathbb{S}_6(\mathbb{C}_{1V} \langle i \rangle \mathbb{C}_{2V}), \mathbb{S}_6(\{x : i \cdot b_1 + \bar{i} \cdot b_2\})\big)$$

is defined and most general, and thus ensures that the type of x is well-formed. Hence by the certify rule and the definition of $\mathsf{Constraints}$,

$$\begin{aligned}
&\Big\langle \mathbb{S}_6(\Gamma_V, \Theta_V, x : (\alpha^i)^j), \mathbb{S}_6(\mathcal{C}_1, \mathcal{C}_2, b_1 \leq_{\mathsf{B}} \mathbb{C}_{1\mathrm{T}}(V), b_2 \leq_{\mathsf{B}} \mathbb{C}_{2\mathrm{T}}(V)) \\
&\qquad\qquad \mathbb{S}_6(\mathsf{Proc}^{i \cdot b_1 + \bar{i} \cdot b_2}), \mathbb{S}_6(\mathbb{C}_{1V} \langle j \rangle \mathbb{C}_{2V}, x : i \cdot b_1 + \bar{i} \cdot b_2) \Big\rangle \\
&\leq \\
&\Big\langle (\Gamma', \Theta', x : (\sigma^i)^d, \Delta), (\mathcal{C}'_1, \mathcal{C}'_2, c_1 \leq_{\mathsf{B}} \mathbb{C}'_{1\mathrm{T}}(V), c_2 \leq_{\mathsf{B}} \mathbb{C}'_{2\mathrm{T}}(V)) \\
&\qquad\qquad \mathsf{Proc}^{i \cdot c_1 + \bar{i} \cdot c_2}, (\mathbb{C}'_{1V} \langle i \rangle \mathbb{C}'_{2V}, x : (i \cdot c_1 + \bar{i} \cdot c_2)) \Big\rangle
\end{aligned}$$

as required.

□

5.5.3 Termination of $\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}$

Theorem 5.5.10 *The algorithm* $\mathsf{Type}'^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}$ *terminates for all finite inputs.*

Proof. By induction on the length of the input. For unary input, the algorithm is defined as a call to $\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}$ which terminates by Theorem 5.5.11. For a sequence, the algorithm is defined as a call to $\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}$ which terminates as above, and a recursive call on a subset of the input, which terminates by the induction hypothesis. The respective outputs are unified with calls to Unify and CUnify, which terminate via Theorems 5.2.15 and 5.2.18. Hence since each stage either terminates or makes a recursive call on a subset of the input, if the input is finite the algorithm must eventually terminate. □

Theorem 5.5.11 *The algorithm* $\mathsf{Type}^{\mathbb{C}_\epsilon}_{\mathcal{T}_{\mathsf{HO}\pi\leq}}$ *terminates for all finite inputs.*

Proof. By induction on the structure of the input. Each step either: terminates immediately (returning a result); calls one of the algorithms Unify or $\mathbb{M}$ which terminates by Theorems 5.2.15 and 5.2.21; or makes a recursive call on a subset of the input. Since each stage either terminates explicitly or recurses on a subset of the input, if the input is finite then the algorithm must eventually terminate.

For instance; the case for a certify construct begins with a recursive call to each branch, and by the induction hypothesis these both terminate. This is followed by three separate calls to Unify and one each to $\mathbb{M}$ and CUnify, which all terminate by Theorems 5.2.15, 5.2.18, and 5.2.21 respectively. Thus the algorithm terminates for all certify constructs containing finite sub-terms. □

5.6 Discussion

This Chapter discussed implementations, in the form of type inference algorithms, for Systems $\mathcal{T}_{\mathsf{FO}\pi}$, $\mathcal{T}_{\mathsf{FO}'\pi\leq}$, and $\mathcal{T}_{\mathsf{HO}\pi\leq}$. Correctness was demonstrated by proofs of soundness (that a valid deduction could be formed from the information returned by the algorithm) and completeness (that for any valid derivation, the algorithm was defined for the corresponding term and produced a most-general result).

For all systems, a substitution was returned in addition to a derived environment and other relevant information. This was to handle the possibility of variable annotations in declarations being unified and not matching types in the environment. Proofs of soundness were then couched in terms of this substitution.

For the latter two systems involving sub-typing, a set of constraints on annotations was also returned, as there is by definition a range of possible solutions for a term. These constraint sets were also considered in the most-general determination, by comparing against a set of of constraints derived from a derivation tree.

Chapter 6

Future Work and Conclusions

This chapter summarises the results of the thesis, and considers future directions for enhancement.

6.1 Future Work

6.1.1 A More Powerful Subtyping Discipline

In Section 3.4.1 it was observed that complete *structural* sub-typing (that is, considering more than just the annotation on the bare type, as in the systems introduced here) by necessity requires a further notion of the input/output usage of that channel; as first introduced in Pierce and Sangiorgi (1993) and further analysed (with the initial work on inference algorithms for such systems) in Igarashi and Kobayashi (2000).

In that context, a natural extension of the work presented in this thesis would be including such information, to enable a deeper (and, it would appear, more useful) notion of sub-typing. The logical method of incorporating I/O information would appear to be to simply add an additional tag representing I/O usage; so an annotation now becomes a pair of an integrity expression and a I/O usage tag, for example b/o (where o is either $+$, $-$, or $\pm$ to indicate output only, input only, or both respectively). This is consistent with the scheme in Pierce and Sangiorgi (1993) — which is concerned solely with sub-typing based on I/O information — but may appear unusual in comparison with systems such as those detailed in (for example) Kobayashi et al. (1996): many similar systems utilise a *pair* of tags, one expression describing properties when the name is used for input, and the other when used for output.

There are two reasons for rejecting such a scheme in this context:

- It is highly unlikely that the integrity of a channel changes dependent on the direction of its use.
- The schemes mentioned above are concerned with *usage analysis* of channels, in particular on those cited with linearity properties; in that setting it is natural to track input and output usage, as the semantics of reduction in the π-calculus necessitate that a channel must be used at least once in both an input and output setting. No such constraint exists in the scenarios examined here.

With these assumptions, a tentative type system — extending that of Figure 3.4 on page 57 — may be proposed.

Definition 6.1.1 (Input/Output Annotations) *Let o, p, q range over I/O annotations, with constants $+$, $-$, and $\pm$, where $+$ represents output only, $-$ input only, and $\pm$ both input and output (see Pierce and Sangiorgi 1993). The relation $\leq_I$ is the least transitive and reflexive relation closed under*

$$\pm \leq_I + \qquad\qquad \pm \leq_I -$$

Definition 6.1.2 (Type Syntax) *The bare type syntax is now:*

$$\sigma \quad ::= \quad \left(\overrightarrow{\sigma^{b/o}}\right)$$

where b is a trust expression as before (see Definition 3.1.1), and o is an I/O annotation as in Definition 6.1.1. Where convenient, the symbols m, n will range over the combined annotations b/o. Multiplication of types by a trust expression is as before, applying only to the trust expression part of the annotation: $b \cdot \sigma^{c/o} = \sigma^{bc/o}$ (also extended point-wise to multiplication of sequences as before).

Definition 6.1.3 (Subtype Relation) *The relation $\leq$ is the least transitive and reflexive relation containing*

$$\frac{o \leq_I - \quad b \leq_{\mathsf{B}} c \quad \overrightarrow{\sigma^m} \leq \overrightarrow{\rho^n}}{(\overrightarrow{\sigma^m})^{b/o} \leq (\overrightarrow{\rho^n})^{c/-}} \qquad \frac{o \leq_I + \quad b \leq_{\mathsf{B}} c \quad \overrightarrow{\sigma^m} \leq \overrightarrow{\rho^n}}{(\overrightarrow{\sigma^n})^{b/o} \leq (\overrightarrow{\rho^m})^{c/+}}$$

$$\frac{b \leq_{\mathsf{B}} c \quad \overrightarrow{\sigma^m} = \overrightarrow{\rho^n}}{(\overrightarrow{\sigma^m})^{b/\pm} \leq (\overrightarrow{\rho^n})^{c/\pm}}$$

Note the contravariant ordering in the output sub-type case on the right; this expresses the same idea as in the output rule of Figures 3.4 and 3.7.

$$\frac{\vdash_\pi \overrightarrow{\sigma^{b/m}}}{\vdash_\pi (c \cdot \overrightarrow{\sigma^{b/m}})^{c/n}}\ (type.) \qquad \frac{\Gamma \vdash^b_{(\pi)} \mathrm{P} : \mathsf{Proc}}{\Gamma \vdash^b_\pi \mathrm{P} : \mathsf{Proc}}\ (conv.)$$

$$\frac{}{\emptyset \vdash^b_{(\pi)} \mathbf{0} : \mathsf{Proc}}\ (zero) \qquad \frac{\Gamma \vdash^b_\pi \mathrm{P} : \mathsf{Proc} \quad c \leq_{\mathsf{B}} b}{\Gamma \vdash^c_\pi \mathrm{P} : \mathsf{Proc}}\ (rel.)$$

$$\frac{\Gamma \vdash^b_\pi \mathrm{P} : \mathsf{Proc}}{\Gamma \vdash^b_\pi !\mathrm{P} : \mathsf{Proc}}\ (rep.) \qquad \frac{\Gamma_x, x : \sigma^m \vdash^c_\pi \mathrm{P} : \mathsf{Proc}}{\Gamma_x \vdash^c_\pi (\nu x : \sigma^m)\mathrm{P} : \mathsf{Proc}}\ (res.)$$

$$\frac{\Gamma \vdash^b_\pi \mathrm{P} : \mathsf{Proc} \quad x \notin \mathrm{dom}\Gamma \quad \vdash_\pi \sigma^m}{\Gamma, x : \sigma^m \vdash^b_\pi \mathrm{P} : \mathsf{Proc}}\ (weak.)$$

$$\frac{\Gamma \vdash^b_{(\pi)} \mathrm{P} : \mathsf{Proc} \quad \Theta \vdash^c_{(\pi)} \mathrm{Q} : \mathsf{Proc}}{\Gamma, \Theta \vdash^{bc}_{(\pi)} \mathrm{P} + \mathrm{Q} : \mathsf{Proc}}\ (sum.)$$

$$\frac{\Gamma \vdash^b_\pi \mathrm{P} : \mathsf{Proc} \quad \Theta \vdash^c_\pi \mathrm{Q} : \mathsf{Proc}}{\Gamma, \Theta \vdash^{bc}_\pi \mathrm{P}|\mathrm{Q} : \mathsf{Proc}}\ (comp.)$$

$$\frac{\Gamma_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^m})^{c/o}, \overrightarrow{y} : \overrightarrow{\sigma'^n} \vdash^d_\pi \mathrm{P} : \mathsf{Proc} \quad \overrightarrow{\sigma^m} \leq \overrightarrow{\sigma'^n} \quad d \leq_{\mathsf{B}} c}{\Gamma_{\overrightarrow{y}}, x : (\overrightarrow{\sigma^m})^{c/} \vdash^d_{(\pi)} x(\overrightarrow{y} : \overrightarrow{\sigma'^n}).\mathrm{P} : \mathsf{Proc}}\ (inp.)$$

$$\frac{\Gamma, x : (\overrightarrow{\sigma^m})^{c/o}, \overrightarrow{y} : \overrightarrow{\sigma'^n} \vdash^d_\pi \mathrm{P} : \mathsf{Proc} \quad \overrightarrow{\sigma'^n} \leq \overrightarrow{\sigma^m} \quad d \leq_{\mathsf{B}} c}{\Gamma, x : (\overrightarrow{\sigma^m})^{c/o}, \overrightarrow{y} : \overrightarrow{\sigma'^n} \vdash^d_{(\pi)} \overline{x}[\overrightarrow{y}].\mathrm{P} : \mathsf{Proc}}\ (out.)$$

$$\frac{\begin{array}{ll}\Gamma_y, x : (\sigma^{i/p})^{b/o}, y : \sigma^{\mathrm{T}/p} \vdash^c_\pi \mathrm{P} : \mathsf{Proc} & c \leq_{\mathsf{B}} b \\ \Theta_y, x : (\sigma^{i/p})^{b/o}, y : \sigma^{\mathrm{U}/p} \vdash^d_\pi \mathrm{Q} : \mathsf{Proc} & d \leq_{\mathsf{B}} b\end{array}}{\Gamma_y, \Theta_y, x : (\sigma^{i/p})^{b/o} \vdash^{ic+\bar{i}d}_{(\pi)} x(y : \sigma^{i/p})_{\underset{\text{certify}}{?}}\mathrm{P} \oplus \mathrm{Q} : \mathsf{Proc}}\ (cert.)$$

Figure 6.1: Proposed Extended First-Order I/O Sub-Type Rules

The (proposed) new rules for the first-order calculus are displayed in Figure 6.1.

There is scope for incorporating aspects of I/O information in the higher-order system as well, by considering the direction of the ports in the external context. As an example, consider the process (assumed untrusted)

$$\overline{x}.\mathrm{P}$$

where the external context contains just x. In this example, the process clearly shares a name with the host; however if the host only uses x for outward communication then calling the above process untrusted (after the final rule) is perhaps unjust.

To fine tune this, perhaps let the external context be a set of names and I/O tags; for example $\mathbb{C}_\epsilon = \{x^-\}$. A context is defined similarly; for example $\mathbb{C} = \{x^+ : \mathrm{U}\}$. Also define $\overline{x^-} = x^+$ and $\overline{x^+} = x^-$, then the final rule annotation could be adjusted to be $\bigwedge_{x^o \in \mathbb{C}_\epsilon} \mathbb{C}_\mathrm{T}(\overline{x^o})$, and the above example would thus be trusted after the final rule (this in fact resembles the filters of Vivas and Yoshida (2002)).

6.1.2 A Deeper Structure for Contexts

The system $\mathcal{T}_{\mathsf{HO}\pi\leq}$ used contexts which had an essentially flat structure. This served to implement the observability-based security property beind the intuitions, but also proved a little limiting in some circumstances, notable agent subtyping. It would seem that in order to use similar ideas in conjunction with subtyping a more flexible or powerful structure is needed.

Some preliminary investigations have been done in to a system of context combinators that reflect the structure of the term they relate to. For example, in a composition rather than just set union they might be combined as $\mathbb{C}_1|\mathbb{C}_2$, for some definition of '|' that is at least associative and symmetric. This is a rather complicated direction, but the notion of graph types (Yoshida 1996) may offer some useful insights.

6.1.3 Static Optimisation

There may well be situations in which the certify input has statically determinable integrity — for instance, if it is received on an untrusted channel then it must itself be untrusted, and hence the second branch of a certify expression will *always* be taken (the semantics of coercion only permit coercion of variable trustedness; it is illegal for a trusted value to be coerced to untrusted, and vice-versa) and the first branch is redundant. This opens up a trivial set of optimisations along the lines of dead code removal.

6.1.4 Extending the Properties of Certify

One of the novel contributions presented in this thesis has been the concept of certify; an oracle-nature verification procedure capable of determining integrity at run-time. This is not an unrealistic concept; many examples of such a procedure were proffered, such as digital signature verification, checksums (up to a certain level of noise), proof-carrying code, and so on. It does, however, introduce a certain representational gap; most "real world" systems utilising such procedures most likely use a combination of many.

This introduces the possibility of extending the system with different classes of certify procedures, each capable of handling different types of request. Aside from being a more accurate model of many situations, it opens the possibility of a finer-grained type system: types may be tagged not just with an integrity expression, but with a value describing perhaps the checks already passed. For example, in a two-tiered authentication system — in which the first test must be passed before the second is considered — any data path inadvertently leading to entry into the second class of certify bypassing the first would be detected as not carrying the correct tag.

The second possibility for extending certify would be to introduce a *range* of values it can classify data at (for example, untrusted/not-sure/trusted). This is similar to a lattice-based approach, but could conceivably be more easily adapted to the systems here using a *fuzzy logic* instead of the current crisp system. See Section 6.1.6 for further ideas in this direction.

6.1.5 Generalising the rules of $\mathcal{T}_{\mathsf{HO}\pi\leq}$

A weakness of the rules of $\mathcal{T}_{\mathsf{HO}\pi\leq}$ is the need to know the execution environment *in advance*; that is, the rules are with respect to $\mathbb{C}_\epsilon$ and if a client wishes to check the program with respect to a different environment they must perform the entire deduction from scratch. It would be desirable if this restriction could be factored out in some manner, so the final result is an expression dependent on $\mathbb{C}_\epsilon$ (rather than derived from it) and the result may be checked for different environments simply by plugging in different values of $\mathbb{C}_\epsilon$.

It is presently unclear how this might proceed, however systems based on proof-carrying code (Necula 1998; Necula and Lee 1998) would appear to be a good starting point for analysis.

6.1.6 Generalising Annotation Semantics

The flexibility and power of boolean-style type annotations has seen a rapid adoption of their use in a variety of analyses. In fact it would appear possible that a general framework could be constructed around that presented here, capable of handling a much wider range of analyses than just integrity, simply by changing the interpretation of the constant symbols. This idea was first presented in Wright (1996) using logics as the basis (with examples showing how the same rules could express several usage analyses), and similar work has since been attempted using the π-calculus as the modelling tool (for example Igarashi and Kobayashi 2001).

6.1.7 A Different Modelling Language

The π-calculus is a powerful tool for reasoning about concurrency, and served this purpose well in this thesis, however it also exposed several short-comings. Its main drawback (which is also, it should be noted, a desirable feature in many circumstances, such as when only the minimal abstraction of concurrency is of concern) is the lack of any notion of *locality*. Attempts have been made to extend the π-calculus with locality primitives (for example Sewell 1998), however it is proposed that a cleaner syntax and corresponding analysis may be obtained by using a modelling language designed for that purpose from the start.

One such calculus seemingly ideally suited for such a task is the ambient calculus (Cardelli and Gordon 1998). In this calculus, locality is primitive and easily syntactically expressed; locations may be nested (representing, for example, different levels of firewall protection within an organisation), and boundaries may be opened or closed. Work on security properties for ambients has naturally already been undertaken (for example Bugliesi and Castagna 2000; Bodei and Levi 2000), and it would be intriguing to see if the same analyses described in this thesis (including boolean annotations to express integrity properties and a certify-like run-time verification procedure) can be easily applied to the ambient calculus. Several variants on the ambient calculus also exist, many designed with security in mind, such as for example the Seal calculus (Castagna, Vitek and Nardelli 2003).

An alternative language is the box-π calculus (Sewell and Vitek 1999); an aesthetically-pleasing combination of ambient-style boundaries and π-calculus channels for communication, which may even provide a closer mapping of the concepts here.

Lastly, the join-calculus (Fournet and Gonthier 1996) is another promising language, offering both a powerful set of abstractions such as those provided

by the π-calculus and a close mapping with an ML-like language (Fournet and Gonthier 1996; Fournet, Laneve, Maranget and Rémy 1997).

6.1.8 Protocol Analysis

Abadi and Gordon (1997) and Abadi, Fournet and Gonthier (1998) described a variant of the π-calculus (enriched with operators for describing cryptographic primitives, such as keys and encryption) that was useful for reasoning about cryptographic protocols in a formal manner. Gordon and Jeffrey (2001) extend this notion and present a practical tool implementing the analysis. It would be interesting to see if the analyses and methods of this thesis can perform a similar function, and what role certify can play.

6.2 Conclusions

This thesis has described a system and analysis for statically determining the safety of certain types of run-time coercion. The π-calculus was used as the modelling language, and a minimal syntactic extension introduced to facilitate the analysis. Analysis was done via type systems, using an algebra of boolean annotations to represent the integrity of each variable.

Chapter 3 presented four such systems:

- System $\mathcal{T}_{\mathsf{FO}\pi}$, an annotated type system for the first-order π-calculus.
- System $\mathcal{T}_{\mathsf{FO}\pi\leq}$, an annotated type system for the first-order π-calculus with sub-typing. Sub-Typing allowed more programs to be typed, by allowing trusted data to be written to untrusted channels.
- System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$, an extension of the previous system that annotates each derivation with a security level, in order to guarantee that untrusted channels cannot affect the reduction of a process typed at a trusted level.
- System $\mathcal{T}_{\mathsf{HO}\pi\leq}$, an annotated type system for the higher-order π-calculus with channel sub-typing. This allowed the analysis of systems incorporating mobile code. As processes themselves now become first-class data objects, it is natural that they are also assigned integrity levels. This led to a conundrum as to how to assign a trustedness level to a group of processes. The solution was to perform the deduction from the perspective of the environment hosting the processes, and the individual process' ability to interact with it. A novel concept of *execution context* was introduced in order to perform this analysis.

The cornerstone of all three systems was a novel construct called *certify*. This is a minimalist syntactic addition to the language that makes it possible to perform coercion, including from a lower level to trusted, in an implicit fashion that may be statically verified for safety. It functions by unifying the operations of verification and branching based on the result; it is thus known at compile-time that one branch will always be taken if the verification procedure succeeds, and the other if it fails. It is therefore possible to parameterise the branches, one on a trusted variable and the other on an untrusted variable, and in this manner coercion is available to the programmer but its incorrect or indiscriminate usage is not. In the higher-order calculus the use of boolean constraints enables a precise annotation to be formed for the construct, using the results of verification. The verification procedure itself is kept abstract, that the analysis may become a framework capable of utilising any suitable procedure.

Chapter 4 examined the safety of three of the four the systems (System $\mathcal{T}_{\mathsf{FO}\pi\leq}$ was excluded as it was essentially developed as an intermediate step to presenting System $\mathcal{T}_{\mathsf{FO}'\pi\leq}$). This was achieved in two steps: a demonstration of type *soundness*, that typing is preserved under reduction (that is, subject reduction), and a security property. Subject reduction is complicated by the possibility of coercion: this was solved by stating that types are preserved, modulo a substitution on annotations that is precisely defined by the reduction path (that is, it encapsulates exactly each instance of coercion occurring). The security property for the second first-order system was presented as a form of Secure Non-deterministic Non-Interference, such that no channel of an integrity lower than the deduction is typed at can affect a reduction. In the higher order calculus it was stated that if a group of processes is typed as being trusted, then it would never be the case that a sub-process typed as untrusted be able to communicate with the external environment, in other words as a property based on observability. No security property was presented for System $\mathcal{T}_{\mathsf{FO}\pi}$ as it is not a security system.

Chapter 5 described implementations for each system. This was expressed in terms of type inference algorithms that returned principle types for each valid term. Correctness of each algorithm was additionally shown, in terms of their soundness (each algorithm produces a valid typing), and completeness (for each valid term, the algorithm is defined and further returns a most-general typing).

Bibliography

Abadi (1997). Secrecy by typing in security protocols, *TACS: 3rd International Conference on Theoretical Aspects of Computer Software*.

Abadi (1999). Security protocols and specifications, *FOSSACS: International Conference on Foundations of Software Science and Computation Structures*, LNCS.

Abadi, M., Banerjee, A., Heintze, N. and Riecke, J. G. (1999). A core calculus of dependency, *The 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '99)*, Association for Computing Machinery, New York, pp. 147–160.

Abadi, M., Fournet, C. and Gonthier, G. (1998). Secure implementation of channel abstractions, *Proceedings of the Thirteenth Annual IEEE Symposium on Logic in Computer Science*, pp. 105–116.

Abadi, M. and Gordon, A. D. (1997). A calculus for cryptographic protocols: The spi calculus, *Fourth ACM Conference on Computer and Communications Security*, ACM Press, pp. 36–47.

Barendregt, H. (1981). *The Lambda Calculus, Its Syntax and Semantics*, Vol. 103 of *Studies in Logics and the Foundations of Mathematics*, North Holland, Amsterdam, The Netherlands.

Biba, K. (1977). Integrity considerations for secure computer systems, *Technical Report TR-3153*, Mitre, Bedford, MA.

Bodei, C., Degano, P., Nielson, F. and Nielson, H. R. (2001). Static analysis for the π-calculus with applications to security, *Information and Computation* **168**.

Bodei, C. and Levi, F. (2000). Security analysis for mobile ambients.

Boole, G. (1847). *The Mathematical Analysis of Logic: Being an Essay Toward a Calculus of Deductive Reasoning*, Macmillan, Cambridge.

Bugliesi, M. and Castagna, G. (2000). Secure safe ambients and jvm security.

Cardelli, L. (1996). Type systems, *ACM Computing Surveys* **28**(1): 263–264.

Cardelli, L. and Gordon, A. D. (1998). Mobile ambients, *in* M. Nivat (ed.), *Proceedings of the First International Conference on Foundations of Software Science and Computation Structures (FoSSaCS '98), Held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS'98), (Lisbon, Portugal, March/April 1998)*, Vol. 1378 of *lncs*, sv, pp. 140–155.

Castagna, G., Vitek, J. and Nardelli, F. (2003). The seal calculus, *Technical report*, Département d'Informatique, École Normale Supérieure, Paris, France.

Cryptyc: Cryptographic Protocol Type Checker (2006). Accessed: 3rd January, 2006.
URL: *http://www.cryptyc.org/*

Denning, D. E. (1976). A lattice model of information flow, *Communications of the ACM* **19**(5): 236–243.

Denning, D. E. (1999). The limits of formal security models. National Computer Systems Security Award Acceptance Speech.

Distributed.net (2003). Accessed: 27th May, 2004.
URL: *http://www.distributed.net/*

Focardi, R. and Gorrieri, R. (1995). A classification of security properties, *Journal of Computer Security* **3**(1).
URL: *ftp://ftp.cs.unibo.it/pub/TR/UBLCS/ClassificationOfSecurity.ps.gz*

Fournet, C. and Gonthier, G. (1996). The reflexive CHAM and the join-calculus, *Conference Record of POPL '96: The* 23rd *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, St. Petersburg Beach, Florida, pp. 372–385.

Fournet, C., Laneve, C., Maranget, L. and Rémy, D. (1997). Implict typing à la ML for the join-calculus, *Proc. of the 1997 8th International Conference on Concurrency Theory*, Springer-Verlag.

Gordon, A. and Jeffrey, A. (2001). Authenticity by typing for security protocols, *14th IEEE Computer Security Foundations Workshop (CSFW '01)*, IEEE, Washington - Brussels - Tokyo, pp. 145–159.

Gosling, J., Joy, B., Steele, G. and Bracha, G. (2000). *The Java Language Specification Second Edition*, Addison-Wesley, Boston, Mass.

Heintze, N. and Riecke, J. G. (1998). The SLam calculus: Programming with secrecy and integrity, *Conference Record of POPL 98: The 25TH ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Diego, California*, ACM, pp. 365–377.

Hennessy, M. and Riely, J. (2000). Information flow vs. resource access in the asynchronous pi-calculus, *27th International Colloquium on Automata, Languages and Programming (ICALP '2000)*, Lecture Notes in Computer Science, Springer-Verlag, Berlin Germany. A longer version appeared as Computer Science Technical Report 2000:03, School of Cognitive and Computing Sciences, University of Sussex.

Hepburn, M. and Wright, D. (2001). Trust in the pi-calculus, *Third International Conference on Principles and Practice of Declarative Programming (PPDP'01)*.

Hepburn, M. and Wright, D. (2003). Execution contexts for determining trust in a higher-order π-calculus, *Technical report*, School of Computing, University of Tasmania.

Hepburn, M. and Wright, D. (2004). Determining trust in cooperating agent-based systems, *in* M. Negnevitsky (ed.), *Artificial Intelligence in Science and Technology 2004*.

Honda, K., Vasconcelos, V. and Yoshida, N. (2000). Secure information flow as typed process behaviour, *in* G. Smolka (ed.), *Programming Languages and Systems: Proceedings of the 9th European Symposium on Programming (ESOP 2000), Held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS 2000), (Berlin, Germany, March/April 2000)*, Vol. 1782 of *lncs*, sv, pp. 180–199.

Honda, K. and Yoshida, N. (2002). A uniform type structure for secure information flow, *Conference Record of POPL'02: The 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Portland, Oregon, pp. 81–92.

Igarashi, A. and Kobayashi, N. (2000). Type reconstruction for linear pi-calculus with I/O subtyping, *Journal of Information and Computation* **161**(1): 1–44. An extended abstract appeared in the *Proceedings of SAS '97*, LNCS 1302.

Igarashi, A. and Kobayashi, N. (2001). A generic type system for the pi-calculus, *28th Annual Symposium on Principles of Programming Languages (POPL) (London, UK)*, ACM.

Jif: Java + information flow (2005). Accessed: 20th November, 2005.
URL: *http://www.cs.cornell.edu/jif/*

Kirli, Z. (2001). Confined mobile functions, *14th IEEE Computer Security Foundations Workshop (CSFW '01)*, IEEE, Washington - Brussels - Tokyo, pp. 283–294.

Kobayashi, N., Pierce, B. and Turner, D. (1996). Linearity and the pi-calculus.

Martin, U. and Nipkow, T. (1990). Boolean unification — the story so far, *in* C. Kirchner (ed.), *Unification*, Academic Press, pp. 437–456.

Milner, R. (1980). *A Calculus for Communicating Systems*, Vol. 92 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin.

Milner, R. (1992). Functions as processes, *Mathematical Structures in Computer Science* **2**(2): 119–141.

Milner, R. (1993). The polyadic π-calculus: A tutorial, *in* F. L. Hamer, W. Brauer and H. Schwichtenberg (eds), *Logic and Algebra of Specification*, Springer-Verlag.

Milner, R., Parrow, J. and Walker, D. (1992a). A calculus of mobile processes, I, *Information and Computation* **100**(1): 1–40.

Milner, R., Parrow, J. and Walker, D. (1992b). A calculus of mobile processes, II, *Information and Computation* **100**(1): 41–77.

Myers, A. C. (1999). Jflow: practical mostly-static information flow control, *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, ACM Press, pp. 228–241.

Myers, A. C. and Liskov, B. (1997). A decentralized model for information flow control, *Proceedings of the 16th Symposium on Operating Systems Principles (SOSP-97)*, Vol. 31,5 of *Operating Systems Review*, ACM Press, New York, pp. 129–142.

Myers, A. C., Sabelfeld, A. and Zdancewic, S. (2004). Enforcing robust declassification, *CSFW*, pp. 172–186.
URL: *http://doi.ieeecomputersociety.org/10.1109/CSFW.2004.9*

Necula, G. C. (1998). Compiling with proofs, *Technical report*, Carnegie Mellon University.

Necula, G. C. and Lee, P. (1998). Safe, untrusted agents using proof-carrying code, *in* G. Vigna (ed.), *Mobile Agents and Security*, Vol. 1419 of *Lecture Notes in Computer Science*, Springer-Verlag.

Ørbæk, P. (1995). Can you Trust your Data?, *in* M. I. S. P. D. Mosses and M. Nielsen (eds), *Proceedings of the TAPSOFT/FASE'95 Conference*, Vol. 915 of LNCS of *Springer Lecture Notes in Computer Science*, Springer-Verlag, Aarhus, Denmark, pp. 575–590.

Palsberg, J. and Ørbæk, P. (1995). Trust in the λ-calculus, *in* A. Mycroft (ed.), *SAS'95: Static Analysis*, Vol. 983 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 314–330.

Pierce, B. C. (2002). *Types and Programming Languages*, The MIT Press, Cambridge, Massachusetts.

Pierce, B. and Sangiorgi, D. (1993). Typing and subtyping for mobile processes, *Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science*, pp. 376–385.

Pottier, F. (2001). A simple view of type-secure information flow in the pi-calculus.

Pottier, F. and Conchon, S. (2000). Information flow inference for free, *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP'00)*, Montral, Canada, pp. 46–57.

Sangiorgi, D. (1993a). *Expressing Mobility in Process Algebra*, PhD thesis, University of Edinburgh, Edinburgh, United Kingdom.

Sangiorgi, D. (1993b). From π-calculus to Higher-Order π-calculus — and back, *in* M.-C. Gaudel and J.-P. Jouannaud (eds), *Proc. TAPSOFT'93*, Vol. 668 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 151–166.

Schneier, B. (1993). *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, John Wiley & Sons, Inc.

SETI@home: Search for Extraterrestrial Intelligence at home (2003). Accessed: 27th May, 2004.
URL: *http://setiathome.ssl.berkeley.edu/*

Sewell, P. (1998). Global/local subtyping and capability inference for a distributed π-calculus, *Lecture Notes in Computer Science* **1443**: 695–706.

Sewell, P. and Vitek, J. (1999). Secure composition of insecure components, *Trusted objects*, Centre Universitaire d'Informatique, University of Geneva.

Smith, G. and Volpano, D. (1998). Secure information flow in a multi-threaded imperative language, *in* ACM (ed.), *Conference record of POPL '98: the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages: papers presented at the Symposium, San Diego, California, 19–21 January 1998*, ACM Press, pp. 355–364.

Vasconcelos, V. T. (1993). A note on a typing system for the higher-order π-calculus.

Vivas, J. L. and Yoshida, N. (2002). Dynamic channel screening in the higher order pi-calculus, *Proceedings of F-WAN, Foundations of Wide Area Network Computing*, Electronic Notes in Theoretical Computer Science, Elsevier Science.

Volpano, D., Smith, G. and Irvine, C. (1996). A sound type system for secure flow analysis, *Journal of Computer Security* **4**(3): 167–187.

Wright, D. A. (1991). A new technique for strictness analysis, *in* S. Abramsky and T. S. E. Maibaum (eds), *TAPSOFT '91: Proceedings of the International Joint Conference on Theory and Practice of Software Development, Volume 2: Advances in Distributed Computing (ADC) and Colloquium on Combining Paradigms for Software Developmemnt (CCPSD)*, Vol. 494 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 235–258.

Wright, D. A. (1992). *Reduction Types and Intensionality in the Lambda-Calculus*, PhD thesis, University of Tasmania.

Wright, D. A. (1996). Linear, strictness and usage logics, *in* M. E. Houle and P. Eades (eds), *Proceedings of Conference on Computing: The Australian Theory Symposium*, Australian Computer Science Communications, Townsville, pp. 73–80.

Yellin, F. (1995). Low level security in Java, *Fourth International Conference on the World-Wide Web*, MIT, Boston.

Yoshida, N. (1996). Graph types for monadic mobile processes, *in* V. Chandru and V. Vinay (eds), *Proceedings of the 16th Conference on Foundations of Software Technology and Theoretical Computer Science*

(Hyderabad, India, December 18–20, 1996), Vol. 1180 of *LNCS*, Springer, pp. 371–386. Full version as Technical Report ECS-LFCS-96-350, University of Edinburgh.
URL: *http://www.dcs.ed.ac.uk/lfcsreps/EXPORT/96/ECS-LFCS-96-350/index.html*

Yoshida, N. and Hennessy, M. (2000). Assigning types to processes, *15th Symposium on Logic in Computer Science (LICS' 00)*, IEEE, Washington - Brussels - Tokyo, pp. 334–348.

Zdancewic, S. and Myers, A. (2001). Robust declassification, *14th IEEE Computer Security Foundations Workshop (CSFW '01)*, IEEE, Washington - Brussels - Tokyo, pp. 15–26.

Zdancewic, S., Zheng, L., Nystrom, N. and Myers, A. C. (2001). Untrusted hosts and confidentiality: Secure program partitioning, *SOSP*, pp. 1–14.
URL: *http://doi.acm.org/10.1145/502034.502036*